(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 April 2001 (19.04.2001)

PCT

(10) International Publication Number
WO 01/27781 A2

(51) International Patent Classification⁷: G06F 15/16

(21) International Application Number: PCT/US00/26728

(22) International Filing Date: 29 September 2000 (29.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

| | | |
|---|---|---|
| 60/159,086 | 13 October 1999 (13.10.1999) | US |
| 09/672,909 | 28 September 2000 (28.09.2000) | US |

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier applications:

| | |
|---|---|
| US | 60/159,086 (CIP) |
| Filed on | 13 October 1999 (13.10.1999) |
| US | 09/672,709 (CIP) |
| Filed on | 28 September 2000 (28.09.2000) |

(71) Applicant *(for all designated States except US)*: **TIMES N SYSTEMS, INC.** [US/US]; Bldg. B, Suite P, 1908 Kramer Lane, Austin, TX 78758 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **BRIDGERS, Vince** [US/US]; 11337 Pebble Garden Lane, Austin, TX 78739 (US).

(74) Agent: **BRUCKNER, John, J.**; Wilson Sonsini Goodrich & Rosati, 650 Page Mill Road, Palo Alto, CA 94304-1050 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

**Published:**
— *Without international search report and to be republished upon receipt of that report.*

WO 01/27781 A2

(54) Title: LOW LATENCY, HIGH BANDWIDTH MULTI-COMPUTER SYSTEM INTERCONNECT

(57) **Abstract:** Methods, systems and devices are described for a low latency, high bandwidth multi-computer system interconnect. A method includes passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including: receiving said set of interconnect fabric data with said shim layer, classifying said set of interconnect fabric data with said shim layer, and handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated. The methods, systems and devices provide advantages because the speed and scalability of parallel processor systems is enhanced.

# LOW LATENCY, HIGH BANDWIDTH MULTI-COMPUTER SYSTEM INTERCONNECT

## REFERENCE TO APPENDIX

An appendix is included in this application by way of attachment, the totality of which is hereby incorporated by reference as an integral part of this application. The appendix includes printed source code that is discussed below in more detail as a nonlimiting example of the invention.

## BACKGROUND OF THE INVENTION

### 1. Field of the Invention

The invention relates generally to the field of computer systems which have multiple processing nodes and in which each processing node is provided with private, local memory and also in which each processing node has access to a range of memory which is shared with other processing nodes. More particularly, the invention relates to computer science techniques that utilize a low latency, high bandwidth multi-computer system interconnect.

### 2. Discussion of the Related Art

The clustering of workstations is a well-known art. In the most common cases, the clustering involves workstations that operate almost totally independently, utilizing the network only to share such services as a printer, license-limited applications, or shared files.

In more-closely-coupled environments, some software packages (such as NQS) allow a cluster of workstations to share work. In such cases the work arrives, typically as batch jobs, at an entry point to the cluster where it is queued and dispatched to the workstations on the basis of load.

In both of these cases, and all other known cases of clustering, the operating system and cluster subsystem are built around the concept of message-passing. The term message-passing means that a given workstation operates on some portion of a job until communications (to send or receive data, typically) with another workstation is necessary. Then, the first workstation prepares and communicates with the other workstation.

Another well-known art is that of clustering processors within a machine, usually called a Massively Parallel Processor or MPP, in which the techniques are essentially identical to those of clustered workstations. Usually, the bandwidth and latency of the interconnect network of an MPP are more highly optimized, but the system operation is the same.

In the general case, the passing of a message is an extremely expensive operation; expensive in the sense that many CPU cycles in the sender and receiver are consumed by the process of sending, receiving, bracketing, verifying, and routing the message, CPU cycles that are therefore not available for other operations. A highly streamlined message-passing subsystem can typically require 10,000 to 20,000 CPU cycles or more.

There are specific cases wherein the passing of a message requires significantly less overhead. However, none of these specific cases is adaptable to a general-purpose computer system.

Message-passing parallel processor systems have been offered commercially for years but have failed to capture significant market share because of poor performance and difficulty of programming for typical parallel applications. Message-passing parallel processor systems do have some advantages. In particular, because they share no resources, message-passing parallel processor systems are easier to provide with high-availability features. What is needed is a better approach to parallel processor systems.

There are alternatives to the passing of messages for closely-coupled cluster work. One such alternative is the use of shared memory for inter-processor communication.

Shared-memory systems, have been much more successful at capturing market share than message-passing systems because of the dramatically superior performance of shared-memory systems, up to about four-processor systems. In Search of Clusters, Gregory F. Pfister 2nd ed. (January 1998) Prentice Hall Computer Books, ISBN: 0138997098 describes a computing system with multiple processing nodes in which each processing node is provided with private, local memory and also has access to a range of memory which is shared with other processing nodes. The disclosure of this publication in its entirety is hereby expressly incorporated herein by reference for the purpose of indicating the background of the invention and illustrating the state of the art.

However, providing high availability for traditional shared-memory systems has proved to be an elusive goal. The nature of these systems, which share all code and all data, including that data which controls the shared operating systems, is incompatible with the separation normally required for high availability. What is needed is an approach to shared-memory systems that improves availability.

Although the use of shared memory for inter-processor communication is a well-known art, prior to the teachings of U.S. Ser. No. 09/273,430, filed March 19, 1999, entitled Shared Memory Apparatus and Method for Multiprocessing Systems, the processors shared a single copy of the operating system. The problem with such systems is that they cannot be efficiently scaled beyond four to eight way systems except in unusual circumstances. All known cases of said unusual circumstances are such that the systems are not good price-performance systems for general-purpose computing.

The entire contents of U.S. Patent Applications 09/273,430, filed March 19, 1999 and PCT/US00/01262, filed January 18, 2000 are hereby expressly incorporated by reference herein for all purposes. U.S. Ser. No. 09/273,430, improved upon the concept of shared memory by teaching the concept which will herein be referred to as a tight cluster. The concept of a tight cluster is that of individual computers, each with its own CPU(s), memory, I/O, and operating system, but for which collection of computers there is a portion of memory which is shared by all the computers and via which they can exchange information. U.S. Ser. No. 09/273,430 describes a system in which each processing node is provided with its own private copy of an operating system and in which the connection to shared memory is via a standard bus. The advantage of a tight cluster in comparison to an SMP is "scalability" which means that a much larger number of computers can be attached together via a tight cluster than an SMP with little loss of processing efficiency.

What is needed are improvements to the concept of the tight cluster. What is also needed is an expansion of the concept of the tight cluster.

## SUMMARY OF THE INVENTION

A goal of the invention is to simultaneously satisfy the above-discussed requirements of improving and expanding the tight cluster concept which, in the case of the prior art, are not satisfied.

One embodiment of the invention is based on a method comprising: passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including: receiving said set of interconnect fabric data with said shim layer, classifying said set of interconnect fabric data with said shim layer, and handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated. Another embodiment of the invention is based on an apparatus, comprising: a shared memory unit; a first system coupled to said shared memory unit; and a second system coupled to said shared memory unit, wherein a data set transfered between said shared memory unit and at least one member selected from the group consisiting of said first system and said second system is received by a shim that is interposed between either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated. Another embodiment of the invention is based on an apparatus comprising: a switch; a first system coupled to said switch; and a second system node coupled to said switch, wherein a data set transfered from said first system to said second system through said switch is received by a shim that is interposed between either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated.

These, and other, aspects of the present invention will be better appreciated and understood when considered in conjunction with the following description and the accompanying drawings. It should be understood, however, that the following description, while indicating preferred embodiments of the present invention and numerous specific details thereof, is given by way of illustration and not of limitation. Many changes and modifications may be made within the scope of the present invention without departing from the spirit thereof, and the invention includes all such modifications.

## BRIEF DESCRIPTION OF THE DRAWINGS

A clear conception of the advantages and features constituting the present invention, and of the components and operation of model systems provided with the present invention, will become more readily apparent by referring to the exemplary, and therefore nonlimiting, embodiments illustrated in the drawings accompanying and forming a part of this specification, wherein like reference numerals designate the same elements. It should be noted that the features illustrated in the drawings are not necessarily drawn to scale.

FIG. 1 illustrates a block schematic diagram of a network, representing an embodiment of the invention.

FIG. 2 illustrates a schematic diagram of a system architecture including a network switch, representing an embodiment of the invention.

FIG. 3 illustrates a block schematic diagram of a system architecture including a dedicated shared memory node device, representing an embodiment of the invention.

FIG. 4 illustrates a block schematic diagram of an interconnect fabric, representing an embodiment of the invention.

## DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention and the various features and advantageous details thereof are explained more fully with reference to the nonlimiting embodiments that are illustrated in the accompanying drawings and detailed in the following description. Descriptions of well known components and processing techniques are omitted so as not to unnecessarily obscure the present invention in detail.

The teachings of U.S. Ser. No. 09/273,430 include a system which is a single entity; one large supercomputer. The invention is also applicable to a cluster of workstations, or even a network.

The invention is applicable to systems of the type of Pfister or the type of U.S. Ser. No. 09/273,430 in which each processing node has its own copy of an operating system. The invention is also applicable to other types of multiple processing node systems; even an interconnect fabric such as, for example, Infiniband.

The invention can be combined with a tight cluster as described in U.S. Ser. No. 09/273,430. A tight cluster is defined as a cluster of workstations or an arrangement within a single, multiple-processor machine in which the processors are connected by a high-speed, low-latency interconnection, and in which some but not all memory is shared among the processors. Within the scope of a given processor, accesses to a first set of ranges of memory addresses will be to local, private memory but accesses to a second set of memory address ranges will be to shared memory. The significant advantage to a tight cluster in comparison to a message-passing cluster is that, assuming the environment has been appropriately established, the exchange of information involves a single STORE instruction by the sending processor and a subsequent single LOAD instruction by the receiving processor.

The establishment of the environment, taught by U.S. Ser. No. 09/273,430 and more fully by companion disclosures (U.S. Provisional Application Ser. No. 60/220,794, filed July 26, 2000; U.S. Provisional Application Ser. No. 60/220,748, filed July 26, 2000; WSGR 15245-711; WSGR 15245-712; WSGR 15245-713; WSGR 15245-715; WSGR 15245-716; WSGR 15245-717; WSGR 15245-718; WSGR 15245-719; WSGR 15245-720, the entire contents of all which are hereby expressly incorporated herein by reference for all purposes) can be performed in such a way as to require relatively little system overhead, and to be done once for many, many information exchanges. Therefore, a comparison of 10,000 instructions for message-passing to a pair of instructions for tight-clustering, is valid.

The invention can include systems software to implement a low latency, high bandwidth multi-computer using existing readily commercially available commodity computer hardware and network devices. The invention can include a method to implement system software support for harnessing multiple, independent compute nodes using existing readily commercially available systems and network equipment or an interconnect fabric.

In general, the invention can include the use of a network driver shim between a network driver layer, and a protocol software layer. The shim passes packets from the protocol software layer through to the network driver layer. Similarly, packets received from the network driver layer side are passed up to the protocol software layer.

A particular packet type identification can be used to decide how to handle received packets. As an example, in the case of the TCP/IP protocol, the Ethernet type identifier is 0x80-0x00, and is used by the shim to decide to pass the packet up to the protocol software layer for proper handling. In the case of low-latency packets taught by this invention, the shim can decide how best to handle the packet. The invention can include transformation of a data set. For some cases, the shim can also implement a lightweight protocol in order to recover from errors encountered on the network media (such as CRC errors, hung network controllers, dropped packets, buffer errors, etc.). The advantages of the invention include improved cost/performance over existing proprietary solutions.

The shim can expose an API (application program interface) for transport middle-ware to use in order to transmit packets, obtain information on local and remote multi-computer nodes, to setup packet receive sinks, and to control the lightweight protocol. Fault tolerance can be achieved by ganging multiple network interface cards in a single system, and either duplicating traffic over multiple network interface cards in a single system, or failing over when a failed NIC or system is detected. Fast recovery methods can be implemented by using network cards which give media sense interrupt indications, or by using relatively frequent "heartbeat" packets across the media.

Referring to FIG. 1, the invention can be implemented in the context of a network. A first network device/driver 110 is coupled to a network 100. A first shim 120 is coupled to the first network device/driver 110. A first protocol layer 130 is coupled to the first shim 120. The first shim 120 and the first protocol layer 130 can both interface with a first transport application program interface (API) 135.

Still referring to FIG. 1, a second network device/driver 140 is coupled to the network 100. A second shim 150 is coupled to the second network device/driver 140. A second protocol layer 160 is coupled to the second shim

150. The second shim 150 and the second protocol layer 160 can both interface with a second transport API 165.

The shims 120, 150 permit handling of data (e.g., routing and/or transformation) based on the type of data and/or the type of application associated with the transport APIs 135 and 165. The transport APIs may be for the same, or different, applications.

Referring to FIGS. 2-3, different types of system interconnects may be used. One example is the use of a true peer-to-peer interconnect through a network interconnect fabric (such as network switch). FIG. 2 depicts this arrangement. A system 0, a system 1, a system 2 and a system n-1 are all coupled to a network swich 200. System-to-system communication is accomplished through network communication provided by the network interface cards, media and network communications devices in the network.

Another system architecture that makes use of this capability is comprised of multiple compute nodes interconnected through a dedicated shared memory device. This model utilizes a "load-store" approach to remote memory access rather than message passing. This method reduces the cost associated with using a network communications switching fabric, and provides each system with a low latency, high bandwidth path to memory that is accessible by each compute node present in a particular configuration. An example of such a system structure is depicted in FIG. 3. In this embodiment, the system 0, the system 1, the system 2 and the system n-1 are all coupled to a dedicated shared memory node device 300. The dedicated shared memory node device may be RAM and/or a disk.

The system architecture of the invention may be used to implement any or all of the following subsystems:

1. Network access through shared memory.

2. A shared memory disk, where each system's backing store may be cached, and available in the dedicated shared memory node device.

3. Locking primitives for controlled access to shared regions of memory.

Having a portion of shared memory common to each system allows each of the individual systems to have access to their own memory without the normal overhead of cache coherency mechanisms usually used for tightly-coupled, shared memory multiprocessor systems.

Referring to FIG. 4, the invention can be implemented in the context of an interconnect fabric. A first interconnect fabric interface 410 is coupled to an interconnect fabric 400. A first shim 420 is coupled to the first interconnect fabric interface 410. A first protocol layer 430 is coupled to the first shim 420. The first shim 420 and the first protocol layer 430 can both interface with a first transport application program interface (API) 435.

Still referring to FIG. 4, a second interconnect fabric interface 440 is coupled to the network 400. A second shim 450 is coupled to the second interconnect fabric interface 440. A second protocol layer 460 is coupled to the second shim 450. The second shim 450 and the second protocol layer 460 can both interface with a second transport API 465.

Again, the shims 420, 450 permit handling of data (e.g., routing and/or transformation) based on the type of data and/or the type of application associated with the transport APIs 435 and 465. Again, the transport APIs may be for the same, or different, applications.

The context of the invention can include multi-computing. The context of the invention can include fault tolerance. The context of the invention can include shared-system network access. The context of the invention can include a shared network. The invention can include a network driver shim. The context of the invention can include an interconnect fabric, such as, for example, Infiniband.

The invention is an improvement over current clustering implementations in that traffic is intercepted and acted upon at the network device driver layer, and sent at the network device driver layer, and the invention also allows existing communication protocols to still use the same media. This provides a cost/performance benefit to the end customer.

This invention can be primarily systems software. Hardware accelerations can be applied by selecting network interface cards, which provide programmable packet type identification, and automatic media sense detection indications.

The invention can be implemented in the context of an ethernet network.

The ethernet can be connected to each of a plurality of PC machines by a NIC card (network interface card) inside each PC. A NIC has its own required appllication interface (API). NIC's are intended to pass messages between PC's. These messages tend to be somewhat long and somewhat infrequent, so are not well suited for shared memory, which is why the preferred design does not use NIC's . Additionally, they tend to be very simple, which means that more processing is required in the software.

The invention can include a device driver which presents an API to the OS and also does all of the processing NICs require. The invention can then also present the data to the NIC using its require API (the "transport API"). The invention permit a shared-memory machine to be run over a standard network, albeit slower than the machine disclosed in U.S. Ser. No. 09/273,430. Certain applications may not have many LOADS and STORES to shared memory, in which case they will run about as well over a standard set of PC's with industry standard network interconnections as they will on the hardware disclosed in U.S. Ser. No. 09/273,430.

The invention can also be implemented in the context of an interconnect fabric where a separate processor with some of its own memory is provided on a NIC. An example of an appropriate interconnect fabric is Infiniband. In this way, a much simpler method can be defined by which a main processor, when it needs to send or receive some data, just presents a special, short descriptor to the processor on the NIC and lets this NIC processor actually GET or PUT the data.

While not being limited to any particular performance indicator or diagnostic identifier, preferred embodiments of the invention can be identified one at a time by testing for the substantially highest performance. The test for the substantially highest performance can be carried out without undue experimentation by the use of a simple and conventional benchmark (speed) experiment.

The term substantially, as used herein, is defined as at least approaching a given state (e.g., preferably within 10% of, more preferably within 1% of, and most preferably within 0.1% of). The term coupled, as used herein, is defined as connected, although not necessarily directly, and not necessarily mechanically.

The term means, as used herein, is defined as hardware, firmware and/or software for achieving a result. The term program or phrase computer program, as used herein, is defined as a sequence of instructions designed for execution on a computer system. A program may include a subroutine, a function, a procedure, an object method, an object implementation, an executable application, an applet, a servlet, a source code, an object code, and/or other sequence of instructions designed for execution on a computer system.

EXAMPLE

A specific embodiment of the present invention will now be further described by the following, nonlimiting example which will serve to illustrate in some detail various features of significance. The example is intended merely to facilitate an understanding of ways in which the present invention may be practiced and to further enable those of skill in the art to practice the present invention. Accordingly, the examples should not be construed as limiting the scope of the present invention.

The printed source code attached to this invention disclosure is an example of how this invention would be implemented on Windows NT 4.0 and an Intel or Intel compatible processor based personal computer, using the NDIS intermediate driver model. This example is intended to be exemplary, and does not preclude an implementation on a different system, operating system, or type of network. This example also does not exclude hardware accelerations for network controllers to enhance the capability of that controller for this application. A description of the attached software modules follows (this description is in the order that the files are presented in the appendix):

1. D:\nt4ddk\src\timesn\tnsdrvr\sources - A makefile description for creating the binary image.

2. D:\nt4ddk\src\timesn\tnsdrvr\tnsemul.rc - A file for describing the resource information to be embedded in the binary image.

3. D:\nt4ddk\arc\timesn\tnsdrvr\tnsemul.def - A file for describing the exported functions of the final binary image.

4. D:\nt4ddk\src\times\tnsdrvr\tnsif.h - Describes the constants and structures needed for an application to interface directly with the loaded, executing, binary image.

5. D:\nt4ddk\src\timesn\tnsdrvr\tnsdef.h - Times N Systems Specific macros and constants.

6. D:\nt4ddk\src\timesn\tnsdrvr\tnsdebug.h - Header file for describing function prototypes. Constants, structures, and macros needed for using debug services.

7. D:\nt4ddk\src\timesn\tnsdrvr\tnsapi.h - Header file for describing the exported Times N Systems services for emulating a high-speed interconnect.

8. D:\nt4ddk\src\timesn\tnsdrvr\tns.h - Structures, function prototypes, constants, and macros for the module in whole, including managing the object context, and interfacing to an existing, commodity network interface device.

9. D:\nt4ddk\src\timesn\tnsdrvr\tnsdebug.c - Debug services

10. D:\nt4ddk\src\timesn\tnsdrvr\tnsapi.c - Implementations for the Times N Systems application programming interfaces for an emulated high-speed interconnect.

11. D:\nt4ddk\src\timesn\tnsdrvr\tnsemul.c - Main initialization file, Driver entry, relatively infrequently used functions

12. D:\nt4ddk\src\timesn\tnsdrvr\recv.c - Receive packet processing, including high-speed interconnect transport processing

13. D:\nt4ddk\src\timesn\tnsdrvr\send.c -Send packet processing

1. D:\nt4ddk\src\timesn\tnsclien\tnsclien.h - Client driver header file

2. D:\nt4ddk\src\timesn\tnsclien\tnsclient.c - Client driver implementation (an example of how interconnect transport services would be used).

An experimental system was prototyped using 100Mbit/sec full and half-duplex network equipment, and gave very good throughput numbers.

## Practical Applications of the Invention

A practical application of the invention that has value within the technological arts is waveform transformation. Further, the invention is useful in conjunction with data input and transformation (such as are used for the purpose of speech recognition), or in conjunction with transforming the appearance of a display (such as are used for the purpose of video games), or the like. There are virtually innumerable uses for the invention, all of which need not be detailed here.

## Advantages of the Invention

A system, representing an embodiment of the invention, can be cost effective and advantageous for at least the following reasons. The invention improves the speed of parallel computing systems. The invention improves the scalability of parallel computing systems. The invention improves the overall system throughput for a system multi-computer implementation.

All the disclosed embodiments of the invention described herein can be realized and practiced without undue experimentation. Although the best mode of carrying out the invention contemplated by the inventor is disclosed above, practice of the invention is not limited thereto. Accordingly, it will be appreciated by those skilled in the art that the invention may be practiced otherwise than as specifically described herein.

For example, although the low latency, high bandwidth multi-computer system interconnect described herein can be a separate module, it will be manifest that the low latency, high bandwidth multi-computer system interconnect may be integrated into the system with which it is associated. Furthermore, all the disclosed elements and features of each disclosed embodiment can be combined with, or substituted for, the disclosed elements and features of every other disclosed embodiment except where such elements or features are mutually exclusive.

It will be manifest that various additions, modifications and rearrangements of the features of the invention may be made without deviating from the spirit and scope of the underlying inventive concept. It is intended that the scope of the invention as defined by the appended claims and their equivalents cover all such additions, modifications, and rearrangements.

The appended claims are not to be interpreted as including means-plus-function limitations, unless such a limitation is explicitly recited in a given claim using the phrase "means for." Expedient embodiments of the invention are differentiated by the appended subclaims.

Appendix

**File: D:\nt4DDK\src\timesn\tnsdrvr\sources** **Page 1 of 1**

```
!IF 0
Copyright (c) 1989-1993 Microsoft Corporation

Module Name:
    sources.

Abstract:
    This file specifies the target component being built and the list of
    sources files needed to build that component.  Also specifies optional
    compiler switches and libraries that are unique for the component being
    built.
!ENDIF

MAJORCOMP=ntos
MINORCOMP=ndis

TARGETNAME=tnsemul
TARGETTYPE=EXPORT_DRIVER
TARGETPATH=$(BASEDIR)\lib

TARGETLIBS=$(BASEDIR)\lib\*\$(DDKBUILDENV)\ndis.lib

INCLUDES=$(BASEDIR)\inc;$(BASEDIR)\src\network\inc;..\inc

C_DEFINES=$(C_DEFINES) -DNDIS_MINIPORT_DRIVER
C_DEFINES=$(C_DEFINES) -DNDIS40
C_DEFINES=$(C_DEFINES) -DNDIS40_MINIPORT
C_DEFINES=$(C_DEFINES) -DBINARY_COMPATIBLE=0

MSC_WARNING_LEVEL=/W3 /WX

SOURCES=tnsemul.c   \
    recv.c      \
    send.c      \
    tnsapi.c    \
    tnsdebug.c  \
    tnsemul.rc


```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.rc** **Page 1 of 1**

```
#include <windows.h>
#include <ntverp.h>

/*-----------------------------------------------*/
/* the following lines are specific to this file */
/*-----------------------------------------------*/

/* VER_FILETYPE, VER_FILESUBTYPE, VER_FILEDESCRIPTION_STR
 * and VER_INTERNALNAME_STR must be defined before including COMMON.VER
 * The strings don't need a '\0', since common.ver has them.
 */
#define VER_FILETYPE    VFT_DRV
#define VER_FILESUBTYPE VFT2_DRV_NETWORK
#define VER_FILEDESCRIPTION_STR     "Times N Systems Emulation Layer"
#define VER_INTERNALNAME_STR        "TNSEMUL.SYS"

#include "common.ver"

#include "evtmsg.rc"
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.def** **Page 1 of 1**

```
; DEF File for TNSEMUL.SYS

NAME TNSEMUL.SYS

DESCRIPTION 'TNSEMUL.SYS'

EXPORTS
```

**Fil : D:\nt4DDK\src\timesn\tnsdrvr\tnsif.h** **Page 1 of 1**

```
//*****************************************************************
//
// COPYRIGHT:
//      This program is an unpublished work fully protected by the United
//      States copyright laws and is considered a trade secret belonging to
//      Times N Systems, Inc. To the extent that this work may be
//      considered "published," the following notice applies "© 1999, Times N
//      Systems, Inc." Any unauthorized use, reproduction, distribution,
//      display, modification, or disclosure of this program is strictly
//      prohibited.
//
//*****************************************************************
//--
//*****************************************************************
// Module:
// tnsif.h - Times N Protocol interface constants and structures
//
// Description:
//
// Environment:
// [illegible]
//
// Exports:
//      See Module functions generated by script processing.
//
// Author:
//      Vince Bridgers
//      vinceb@timesn.com
//
//--
//*****************************************************************
#ifndef _TNSIF_H_
#define _TNSIF_H_

//
// Debug Levels
//
#define DEBUG_INFO       0
#define DEBUG_MESSAGE    1
#define DEBUG_WARNING    2
#define DEBUG_VERBOSE    3
#define DEBUG_ERROR      4

//
// Debug mask bit definitions. These are implemented as a bit mask
// and are used to selectivly enable/disable certain classes of debug
// messages.
//
#define DEBUG_MASKEN_ERROR       0x01
#define DEBUG_MASKEN_RECV        0x02
#define DEBUG_MASKEN_SEND        0x04
#define DEBUG_MASKEN_INIT        0x08
#define DEBUG_MASKEN_PACKETDUMP  0x10
#define DEBUG_MASKEN_ENTRYEXIT   0x20


#define FILE_DEVICE_TNS   0x00008301
#define TNS_IOCTL_BASE   0x830
#define IOCTL_TNS_SETDEBUGINFO    CTL_CODE(FILE_DEVICE_TNS,     \
                                    TNS_IOCTL_BASE+0,         \
                                    METHOD_BUFFERED,          \
                                    (FILE_READ_ACCESS | FILE_WRITE_ACCESS))

typedef struct _TNS_IOCTLPACKET {
    ULONG   DebugLevel;
    ULONG   DebugMask;
    ULONG   DebugBreakFlag;
} TNS_IOCTLPACKET, *pTNS_IOCTLPACKET;


#endif // _TNSIF_H_
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdefs.h Page 1 of 2

```
//=========================================================================
//
// COPYRIGHT:
//    This program is an unpublished work fully protected by the United
//    States copyright laws and is considered a trade secret belonging to
//    Times N Systems, Inc. To the extent that this work may be
//    considered "published," the following notice applies: (C) 1999, Times N
//    Systems, Inc. Any unauthorized use, reproduction, distribution,
//    display, modification, or disclosure of this program is strictly
//    prohibited.
//
//==========================================================================
//-*
//=========================================================================
// Module:
//
// Description:
//
// Environment:
// [illegible]
//
// Exports:
//
// Author:
//    Vince Bridgers
//    vinceb@timesn.com
//
//-*
//==========================================================================
typedef LONG          TNS_STATUS;
typedef TNS_STATUS *PTNS_STATUS;

typedef LONG          LOCKID;
typedef LOCKID        *PLOCKID;

typedef LONG          LOCKSTATUS;
typedef LOCKSTATUS  *PLOCKSTATUS;

typedef LONG          TNSKEY;
typedef TNSKEY        *PTNSKEY;

typedef LONG          TNSCPUID;
typedef TNSCPUID      *PTNSCPUID;

typedef LONG          TNSNOTIFYSTATUS;
typedef TNSNOTIFYSTATUS *PTNSNOTIFYSTATUS;


typedef LONG          TNSCOUNTER;
typedef TNSCOUNTER  *PTNSCOUNTER;

typedef LONG          TNSQUEUE;
typedef TNSQUEUE   *PTNSQUEUE;

typedef LONG               TNSQUEUEINFO;
typedef TNSQUEUEINFO      *PTNSQUEUEINFO;

typedef LONG      TNSMEMSIZE;

typedef LONG      TNSMEMFLAGS;


#define NTSTATUS_CUSTOMER_CODE  0x20000000

#define TNS_STATUS_CODE(Severity, StatusCode) (\
     (NTSTATUS_CUSTOMER_CODE | (Severity << 30) | StatusCode))



//
// [illegible] Status [illegible] status
//

typedef enum {
     TNS_SUCCESS=0,
     TNS_NOT_IMPLEMENTED,
};

#define TNS_STATUS_SUCCESS TNS_STATUS_CODE(STATUS_SEVERITY_SUCCESS,     TNS_SUCCESS)

#define TNS_STATUS_NOT_IMPLEMENTED TNS_STATUS_CODE(STATUS_SEVERITY_ERROR,       TNS_NOT_IMPLEMENTED)

```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdefs.h Page 2 of 2

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.h** **Page 1 of 2**

```
//*************************************************************************
//
//COPYRIGHT:
//    This program is an unpublished work fully protected by the United
//    States copyright laws and is considered a trade secret belonging to
//    Times N Systems, Inc.  To the extent that this work may be
//    considered "published," the following notice applies: "1999, Times N
//    Systems, Inc."  Any unauthorized use, reproduction, distribution,
//    display, modification, or disclosure of this program is strictly
//    prohibited.
//
//*************************************************************************
//--
//*************************************************************************
// Module:
//    tnsdebug.h - Times N Protocol debug support functions and the like
//
// Description:
//
// Environment:
//    [illegible]
//
// Exports:
//       See Modular functions generated by script processing.
//
// Author:
//       Vince Bridgers
//       vinceb@timesn.com
//
//==
//*************************************************************************
#ifndef _TNSDEBUG_H_
#define _TNSDEBUG_H_

//
// TNS function to make a beep, useful for debugging occasionally
//
void
TNSMakeBeep(void);


#include "tnsif.h"

//*************************************************************************
// Example of how to use for compile time reminders:
//
// #pragma message(REMIND("put stuff here"))
//
//*************************************************************************
#define DEBUG_QUOTE(x) #x
#define DEBUG_QQUOTE(y) DEBUG_QUOTE(y)
#define REMIND(sz) __FILE__"("DEBUG_QQUOTE(__LINE__)"):"sz

#ifdef DBG

char *GetNDISOidString(NDIS_OID NdisOID, PULONG pFoundFlag);
char *GetNDISStatusString(NDIS_STATUS Status, PULONG pFoundFlag);
char *GetNDISEventString(NDIS_ERROR_CODE ErrorCode, PULONG pFoundFlag);

    VOID
    NdisDumpPacket(
        PNDIS_PACKET Packet);

    #define STATIC

    VOID
    DebugPrint(
        ULONG DebugPrintLevel,
        PCSZ DebugMessage,
        ...
    );

    VOID
    MaskDebugPrint(
        ULONG DebugPrintLevel,
        ULONG DebugPrintMask,
        PCSZ DebugMessage,
        ...
    );

    extern ULONG _gDebugPrintLevel;
    extern ULONG _gDebugPrintMask;
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.h** **Page 2 of 2**

```
    extern ULONG _gDebugBreakFlag;

    #define DEBUG_MODULE "DEBUG: "

    #define DINFO(x, y) \
    DebugPrint(x, "%s", DEBUG_MODULE); \
    DebugPrint(x, "File => %s: ", __FILE__); \
    DebugPrint(x, "Line => %d: ", __LINE__); \
    DebugPrint y;

    #define D(x)  DebugPrint x;

    #define DM(x) MaskDebugPrint x;

    #define DUMP_PACKET(x) NdisDumpPacket(x)

    #define INT3 { _asm int 3 }

    #define BreakPoint() \
    { DbgPrint("Debug Break in file => %s, at line %d\n", __FILE__, __LINE__); \
      if (_gDebugBreakFlag) { _asm int 3 } ; }

    #define MyAssert(c) if (!(c)) {\
    { DbgPrint("Assertion failure: Debug Break in file => %s, at line %d\n", __FILE__, __LINE__); \
      if (_gDebugBreakFlag) { _asm int 3 } ; } }

#else // _DBG

    #define STATIC static
    #define DINFO(x,y)
    #define D(x)
    #define DM(x)
    #define BreakPoint()
    #define INT3
    #define MyAssert(c)
    #define DUMP_PACKET(x)

#endif // _DBG
#endif // _TNSDEBUG_H_
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.h Page 1 of 11

```
[redacted]

#define DECLSPEC_EXPORT _declspec(dllexport)

[redacted]
ULONG
DECLSPEC_EXPORT
__TNS_READ_REGISTER_ULONG(
    IN  PVOID   DeviceHandle,
    IN  PULONG  Register);
[redacted]

[redacted]
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_ULONG(
    IN  PVOID   DeviceHandle,
    IN  PULONG  Register,
    IN  ULONG   RegisterData);
[redacted]

[redacted]
USHORT
DECLSPEC_EXPORT
__TNS_READ_REGISTER_USHORT(
    IN  PVOID   DeviceHandle,
    IN  PUSHORT Register);
[redacted]
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.h** **Page 2 of 11**

```
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//@@
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_USHORT(
    IN  PVOID   DeviceHandle,
    IN  PUSHORT Register,
    IN  USHORT  RegisterData);
//
// Description:
//
// Environment:
//
// Return Value:
//   None
//--
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//@@
UCHAR
DECLSPEC_EXPORT
__TNS_READ_REGISTER_UCHAR(
    IN  PVOID   DeviceHandle,
    IN  PUCHAR  Register);
//
// Description:
//
// Environment:
//
// Return Value:
//   None
//--
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//@@
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_UCHAR(
    IN  PVOID   DeviceHandle,
    IN  PUCHAR  Register,
    IN  UCHAR   RegisterData);
//
// Description:
//
// Environment:
//
// Return Value:
//   None
//--
//---------------------------------------------------------------------------



//---------------------------------------------------------------------------
//@@
VOID
DECLSPEC_EXPORT
__TNS_READ_REGISTER_BUFFER_ULONG(
    IN  PVOID   DeviceHandle,
    IN  PULONG  Register,
    IN  PULONG  pulBuffer,
    IN  ULONG   Count);
//
// Description:
//
// Environment:
//
// Return Value:
//   None
//--
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//@@
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_BUFFER_ULONG(
    IN  PVOID   DeviceHandle,
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.h** **Page 3 of 11**

```
    IN  PULONG  Register,
    IN  PULONG  pulBuffer,
    IN  ULONG   Count);
[redacted]

[redacted]
VOID
DECLSPEC_EXPORT
__TNS_READ_REGISTER_BUFFER_USHORT(
    IN  PVOID   DeviceHandle,
    IN  PUSHORT Register,
    IN  PUSHORT pusBuffer,
    IN  ULONG   Count);
[redacted]

[redacted]
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_BUFFER_USHORT(
    IN  PVOID   DeviceHandle,
    IN  PUSHORT Register,
    IN  PUSHORT pusBuffer,
    IN  ULONG   Count);
[redacted]


[redacted]
VOID
DECLSPEC_EXPORT
__TNS_READ_REGISTER_BUFFER_UCHAR(
    IN  PVOID   DeviceHandle,
    IN  PUCHAR  Register,
    IN  PUCHAR  pucBuffer,
    IN  ULONG   Count);
[redacted]

[redacted]
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_BUFFER_UCHAR(
    IN  PVOID   DeviceHandle,
    IN  PUCHAR  Register,
    IN  PUCHAR  pucBuffer,
    IN  ULONG   Count);
[redacted]
```

```
[redacted]

[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSAcquireLockP(
    IN  PVOID    DeviceHandle,
    IN  PLOCKID pLockID);
[redacted]

[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSReleaseLockP(
    IN  PVOID    DeviceHandle,
    IN  PLOCKID pLockID);
[redacted]

[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueryLockP(
    IN  PVOID        DeviceHandle,
    OUT PLOCKSTATUS pLockStatus);
[redacted]



[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSAllocateLockP(
    IN  PVOID   DeviceHandle,
    IN  TNSKEY  Key,
    OUT PLOCKID *pLockID);
[redacted]

[redacted]
TNS_STATUS
```

```
DECLSPEC_EXPORT
__TNSFreeLockP(
    IN  PVOID   DeviceHandle,
    IN  TNSKEY  Key,
    IN  PLOCKID pLockID);
[redacted]

[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSNotifyCPU(
    IN  PVOID       DeviceHandle,
    IN  TNSCPUID    CpuID,
    IN  PVOID       pMessageBuffer,
    IN  ULONG       MessageLength);
[redacted]

[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSNotifyCPUSync(
    IN  PVOID       DeviceHandle,
    IN  TNSCPUID    CpuID,
    IN  PVOID       pMessageBuffer,
    IN  ULONG       MessageLength,
    IN  PVOID       pCallback,
    IN  PVOID       pContext);
[redacted]


[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueryNotifyStatus(
    IN      PVOID                   DeviceHandle,
    IN      TNSCPUID                CpuID,
    IN  OUT PTNSNOTIFYSTATUS        pCpuNotifyInfo);
[redacted]


[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSRegisterNotifyCallback(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pCallBack,
    IN  PVOID       SysParm1,
    IN  PVOID       SysParm2,
    IN  PVOID       SysParm3);
//
// Description:
//
// Environment:
//
// Return Value:
//[illegible]
//--
//[illegible]


//[illegible]
//--
TNS_STATUS
DECLSPEC_EXPORT
__TNSRegisterNotificationCallback(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pCallBack,
    IN  PVOID       SysParm1,
    IN  PVOID       SysParm2,
    IN  PVOID       SysParm3);
//
// Description:
//
// Environment:
//
// Return Value:
//[illegible]
//--
//[illegible]


//[illegible]
//--
TNS_STATUS
DECLSPEC_EXPORT
__TNSDeRegisterNotificationCallback(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pCallBack);
//
// Description:
//
// Environment:
//
// Return Value:
//[illegible]
//--
//[illegible]


//[illegible]
//--
TNSCPUID
DECLSPEC_EXPORT
__TNSWhoAmI(
    IN  PVOID       DeviceHandle);
//
// Description:
//
// Environment:
//
// Return Value:
//[illegible]
//--
//[illegible]

//[illegible]
//--
TNSCOUNTER
DECLSPEC_EXPORT
__TNSReadOrdinalCounter(
    IN  PVOID       DeviceHandle);
//
// Description:
//
// Environment:
//
// Return Value:
//[illegible]
```

```
[redacted]


[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSAllocateSharedMemory(
    IN      PVOID        DeviceHandle,
    IN      TNSKEY       Key,
    IN      TNSMEMFLAGS  Flags,
    IN      TNSMEMSIZE   Size,
    IN OUT  PVOID        *ppBuffer);
[redacted]


[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSFreeSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  TNSKEY      Key,
    IN  PVOID       Ptr,
    IN  TNSMEMSIZE  Size);
[redacted]

[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSReadSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer);
[redacted]


[redacted]
TNS_STATUS
DECLSPEC_EXPORT
__TNSWriteSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer);
[redacted]
```

```

//---------------------------------------------------------------------------
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSDmaReadSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer,
    IN  PVOID       pCallback,
    IN  PVOID       DMAReadCompleteComtext1,
    IN  PVOID       DMAReadCompleteComtext2);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSDmaWriteSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer,
    IN  PVOID       pCallback,
    IN  PVOID       DMAWriteCompleteComtext1,
    IN  PVOID       DMAWriteCompleteComtext2);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSAllocateWorkQueue(
    IN      PVOID       DeviceHandle,
    IN      TNSKEY      Key,
    IN      PULONG      pQueueLength,
    IN OUT  PTNSQUEUE   *ppTNSQueue);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSFreeWorkQueue(
    IN      PVOID       DeviceHandle,
    IN      TNSKEY      Key,
    IN      PTNSQUEUE   pTNSQueue);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//---------------------------------------------------------------------------
```

```

//****************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedEnqueueToDoP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN      PVOID       pItem,
    IN      ULONG       Length);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//****************************************************************


//****************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedDequeueToDoP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN      PVOID       pItem,
    IN      PULONG      pLength);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//****************************************************************

//****************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueryQLengthP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN      PULONG      pLength);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//****************************************************************


//****************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueueHeadP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN OUT  PTNSQUEUE   *ppTNSQueue);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//****************************************************************


//****************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
```

```
__TNSQueueTailP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN OUT  PTNSQUEUE    *ppTNSQueue);
//
// Description:
//
// Environment:
//
// Return Value:
//  None
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueuePayloadP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN      PVOID        pItem,
    IN      PULONG       pLength);
//
// Description:
//
// Environment:
//
// Return Value:
//  None
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueueNextP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN OUT  PTNSQUEUE    *ppTNSQueue);
//
// Description:
//
// Environment:
//
// Return Value:
//  None
//
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
//
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedInsertQueueItemP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN      PTNSQUEUE    pTNSQueueInsert);
//
// Description:
//
// Environment:
//
// Return Value:
//  None
//
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
//
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedDeleteQueueItemP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN      PTNSQUEUE    pTNSQueueDelete);
//
// Description:
```

```
//
// Environment:
//
// Return Value:
// [illegible]
//
//[illegible]

//[illegible]
//
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueueItemInfoP(
    IN      PVOID           DeviceHandle,
    IN      PTNSQUEUE       pTNSQueue,
    IN      PTNSQUEUEINFO   pTNSQueueInfo);
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//
//[illegible]


TNS_STATUS
DECLSPEC_EXPORT
__TNSGetFirstDeviceInstance(
    PVOID   *ppDeviceInstance);

TNS_STATUS
DECLSPEC_EXPORT
__TNSGetNextDeviceInstance(
    PVOID   pDeviceInstance,
    PVOID   *ppDeviceInstance);

```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h** **Page 1 of 11**

```
//****************************************************************************
//
// COPYRIGHT:
//      This program is an unpublished work fully protected by the United
//      States copyright laws and is considered a trade secret belonging to
//      Times N Systems, Inc. To the extent that this work may be
//      considered "published," the following notice applies "© 1999, Times N
//      Systems, Inc." Any unauthorized use, reproduction, distribution,
//      display, modification, or disclosure of this program is strictly
//      prohibited.
//
//*****************************************************************************
//--
//****************************************************************************
// Module:
//   tnspkt.h - Times N Protocol packet definition for emulated subsystem
//
// Description:
//
// Environment:
//
// Exports:
//      See Module Functions generated by script processing.
//
// Author:
//      Vince Bridgers
//      vinceb@timesn.com
//--
//*****************************************************************************

#ifndef _TNS_H_
#define _TNS_H_
#include <ntddk.h>
#include <ndis.h>
#include <ntddndis.h>
#include <tdikrnl.h>
#include "tnsstats.h"

#define MIN_PACKET_POOL_SIZE    0xff
#define MAX_PACKET_POOL_SIZE    0xffff

//
// shutdown mask values
//

#define SHUTDOWN_DEALLOC_PACKET_POOL     0x00000001
#define SHUTDOWN_DEALLOC_LOOKAHEAD_POOL 0x00000002
#define SHUTDOWN_DEALLOC_RESIDUAL_POOL  0x00000004
#define SHUTDOWN_DEINIT_DEV_INSTANCE     0x00000008
#define SHUTDOWN_DELETE_PIPE             0x00000010
#define SHUTDOWN_TERMINATE_WRAPPER       0x00000040
#define SHUTDOWN_DEREGISTER_PROTOCOL     0x00000080
#define SHUTDOWN_DELETE_DEVICE           0x00000100
#define SHUTDOWN_DELETE_SYMLINK          0x00000200

#define READ_HIDDEN_CONFIG( _Field, ParamType ) \
{                                                  \
    ConfigurationInfo->_Field =                    \
        ReadSingleParameter(ConfigHandle,          \
                            Str ## _Field,         \
                            ConfigurationInfo->_Field, \
                            ParamType);            \
}

#define DECLARE_STRING( _str_ ) STATIC WCHAR Str ## _str_[] = L#_str_

#define ETH_ADDRESS_LEN 6

//
// number of characters that are appended to the RegistryPath when constructing
// the miniport device name
//

#define MPNAME_EXTENSION_SIZE   ( 3 * sizeof(WCHAR))


#define MAX_COMPUTER_NAME_SIZE 16

typedef struct _SMNNodeTable {
    int           LocationSet;
    unsigned char TNMacAddress[HARDWARE_ADDRESS_LENGTH];
    unsigned long TNNodeID;
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h** **Page 2 of 11**

```
    unsigned char TNComputerName[MAX_COMPUTER_NAME_SIZE];
} SMNNodeTable, *pSMNNodeTable;

#define MAX_TEAM_NODES  128

//
// per Adapter control block
//
typedef struct _ADAPTER {
    //
    // Required structure member for using DDK provided list management
    // functions
    //
    LIST_ENTRY Linkage;


    BOOLEAN TNSDriverInitialized;

    //
    // Size of this struct, plus allocated strings
    //
    int AdapterStructSize;

    //
    // structure book keeping
    //
    // TNSDeviceName, MPDeviceName - unicode device names for the intermediate
    // and underlying
    // MP device. the buffers for the strings are allocated as part of the adapter
    // structure allocation and are located just after the structure. Buffer size
    // is fixed at DEVNAME_SIZE
    //
    // ShutdownMask - mask of operations to perform during unbinding from lower MP
    //

    NDIS_STRING TNSDeviceName;
    NDIS_STRING MPDeviceName;
    ULONG ShutdownMask;
    ULONG TNSMPState;
    //
    // miniport vars
    //
    // DevInstance - contains the number at the end of the device instance string
    // as in, "ImSampMP3". This is used by MPInitialize to determine which IM
    // device is being initialized. Comparison via device names is not possible
    // since the RTL unicode routines run at lowered IRQL and MPInitialize cannot
    // guarantee that condition.
    //
    // CopyLookaheadData - TRUE if the IM can directly touch the lookahead data
    //
    // TNSNdisHandle - the handle that identifies the IM device to NDIS
    //
    // BlockingEvent - used to synchronize execution of functions that are
    // awaiting completion
    //
    // FinalStatus - holds status returned in completion routine
    //
    // PacketPoolHandle - handle to pool of NDIS_PACKETs used during Send and Packet
    // recieve operations
    //
    // PacketSList - SList of pre-allocated packet structures
    //
    USHORT DevInstance;
    BOOLEAN CopyLookaheadData;
    NDIS_HANDLE TNSNdisHandle;
    NDIS_EVENT BlockingEvent;
    NDIS_STATUS FinalStatus;
    NDIS_HANDLE PacketPoolHandle;

    //
    // Lookahead and residual buffer Pool vars. residual buffer size
    // is same as TotalSize
    //

    ULONG LookaheadBufferSize;
    NDIS_HANDLE LookaheadPoolHandle;

    //
    // Underlying adapter info - handle, type, etc.
    //
    // LowerMPHandle - the binding handle to the underlying MP
    //
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h** **Page 3 of 11**

```
// BindContext - used in BindAdapterHandler and UnbindAdapterHandler
//
// IMBindContext - used for IM when unbinding from IM MP
//
// MediaType - self explanatory I would hope
//
//
// TotalSize - max # of bytes including the header
//

NDIS_HANDLE LowerMPHandle;
UCHAR       LowerMPMacAddress[HARDWARE_ADDRESS_LENGTH];



NDIS_HANDLE BindContext;
NDIS_MEDIUM MediaType;
ULONG LinkSpeed;
ULONG TotalSize;
LIST_ENTRY ClientList;

//----------------------------------------------------------------------
//
// Objects for managaging the client worker thread
//----------------------------------------------------------------------

ULONG        ListEntryItems;

HANDLE       ClientWorkerThreadHandle;
HANDLE       ServerWorkerThreadHandle;
//
// Spinlock for atomic access to queue
//
KSPIN_LOCK   ListEntryPoolLock;
//
// List of requests for server worker thread
//
LIST_ENTRY   WorkerListEntryPool;

//
// Counting semaphore for number of requests on queue
//
KSEMAPHORE   ClientWorkerRequestSemaphore;

//
// Counting semaphore for number of requests on queue
//
KSEMAPHORE   ClientWorkerResponseSemaphore;

//
// Spinlock for atomic access to queue
//
KSPIN_LOCK   ClientWorkerListSpinLock;
//
// List of requests for client worker thread
//
LIST_ENTRY   ClientWorkerList Entry;


//----------------------------------------------------------------------
// Objects for managing the server worker thread
//----------------------------------------------------------------------

//
// Counting semaphore for number of requests on queue
//
KSEMAPHORE   ServerWorkerRequestSemaphore;
//
// Spinlock for atomic access to queue
//
KSPIN_LOCK   ServerWorkerListSpinLock;
//
// List of requests for server worker thread
//
LIST_ENTRY   ServerWorkerListEntry;

UCHAR        SMNMacAddress[HARDWARE_ADDRESS_LENGTH];

//
// [illegible]
//
NDIS_REQUEST Request;
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h Page 4 of 11

```
    PULONG        BytesNeeded;
    PULONG        BytesReadOrWritten;
    BOOLEAN       LocalRequest;


    PVOID         TNSSharedMemoryPtr;
    ULONG         TNSSharedMemorySize;

#define VIRTUAL_MEMORY  1
#define NONPAGED_MEMORY  2

    int           TNSMemoryType;

    ULONG         TNSClientNodeID;

    SMNNodeTable TeamNodeTable[MAX_TEAM_NODES];

    STATISTICS    MyStats;
    MPSTATS       mpStats;

    KSPIN_LOCK    MyStatsLock;

    unsigned char LocalComputerName[MAX_COMPUTER_NAME_SIZE];

    unsigned char SMNMachineName[16];

} ADAPTER, *PADAPTER;

#define MAX_READWRITE_BUFFER_SIZE   1024


[redacted]

#define NdisRequestLocalSetInfo     NdisRequestGeneric1
#define NdisRequestLocalQueryInfo   NdisRequestGeneric2

[redacted]
typedef struct  _TNS_PACKET_CONTEXT {
     PNDIS_PACKET OriginalPacket;
     PNDIS_BUFFER LookaheadBuffer;
     int          SMNEmulationPacket;
} TNS_PACKET_CONTEXT, *PTNS_PACKET_CONTEXT;

#define PACKET_CONTEXT_FROM_PACKET(_pkt ) ((PTNS_PACKET_CONTEXT)((_pkt)->ProtocolReserved))


#define MEDIA_INFO_SIZE      (sizeof( MEDIA_SPECIFIC_INFORMATION ) + sizeof( ULONG ))


[redacted]

typedef struct  _BUFFER_CONTEXT {
     SINGLE_LIST_ENTRY SListEntry;
     PNDIS_BUFFER NdisBuffer;
} BUFFER_CONTEXT, *PBUFFER_CONTEXT;

[redacted]

typedef struct _CONFIG_DATA {
    ULONG PacketPoolSize;
    ULONG DebugLevel;
    ULONG DebugMask;
    ULONG TNSSMNEmulationMode;
} CONFIG_DATA, *PCONFIG_DATA;

//
// values for error log entries
//

#define TNS_ERROR_MISSING_OID                       0x00010000
#define TNS_ERROR_BAD_REGISTRY_DATA                 0x00020000
#define TNS_ERROR_CANT_INITIALIZE_IMSAMP_DEVICE     0x00040000
#define TNS_ERROR_PACKET                            0x00060000
#define TNS_ERROR_PACKET_POOL                       0x00070000
#define TNS_ERROR_LOOKAHEAD_POOL                    0x00080000
#define TNS_ERROR_VM_LOOKAHEAD_BUFFER               0x00090000
#define TNS_ERROR_LOOKAHEAD_BUFFER                  0x000A0000
#define TNS_ERROR_RESIDUAL_POOL                     0x000B0000
#define TNS_ERROR_VM_RESIDUAL_BUFFER                0x000C0000
#define TNS_ERROR_RESIDUAL_BUFFER                   0x000D0000
#define TNS_ERROR_PROTOCOL_INIT                     0X000F0000

// bad registry data indicator

#define TNS_ERROR_INVALID_IMSAMP_MP_INSTANCE        0x00000004

//
// global vars (not based on a device instance)
//
extern ULONG TNSSharedMemoryNodeEmulation;

extern LIST_ENTRY AdapterList;
extern NDIS_SPIN_LOCK AdapterListLock;
extern NDIS_HANDLE ClientProtocolHandle;
extern NDIS_HANDLE MPWrapperHandle;
extern NDIS_HANDLE LMDriverHandle;
extern PDRIVER_OBJECT IMDriverObject;
extern PDEVICE_OBJECT IMDeviceObject;

extern CONFIG_DATA ConfigData;           // pointer to Registry data

extern NDIS_STRING IMSymbolicName;
extern NDIS_STRING IMDriverName;
extern NDIS_STRING IMMPName;


VOID
MPSendPackets(
    IN  NDIS_HANDLE             MiniportAdapterContext,
    IN  PPNDIS_PACKET           PacketArray,
    IN  UINT                    NumberOfPackets);

VOID
CLSendComplete(
    IN  NDIS_HANDLE             ProtocolBindingContext,
    IN  PNDIS_PACKET            Packet,
    IN  NDIS_STATUS             Status);

VOID
PacketCompletion(
    IN PADAPTER Adapter,
    IN PNDIS_PACKET Packet,
    IN NDIS_STATUS Status);

INT
CLReceivePacket(
    IN  NDIS_HANDLE             ProtocolBindingContext,
    IN  PNDIS_PACKET            Packet);

VOID
MPReturnPacket(
    IN  NDIS_HANDLE             MiniportAdapterContext,
    IN  PNDIS_PACKET            Packet);

NDIS_STATUS
CLReceiveIndication(
    IN  NDIS_HANDLE             ProtocolBindingContext,
    IN  NDIS_HANDLE             MacReceiveContext,
    IN  PVOID                   HeaderBuffer,
    IN  UINT                    HeaderBufferSize,
    IN  PVOID                   LookAheadBuffer,
    IN  UINT                        LookaheadBufferSize,
    IN  UINT                        PacketSize);

VOID
CLReceiveComplete(
    IN  NDIS_HANDLE                 ProtocolBindingContext);

NDIS_STATUS
MPTransferData(
    OUT PNDIS_PACKET                Packet,
    OUT PUINT                       BytesTransferred,
    IN  NDIS_HANDLE                 MiniportAdapterContext,
    IN  NDIS_HANDLE                 MiniportReceiveContext,
    IN  UINT                        ByteOffset,
    IN  UINT                        BytesToTransfer);

VOID
CLTransferDataComplete(
    IN  NDIS_HANDLE     ProtocolBindingContext,
    IN  PNDIS_PACKET    pNdisPacket,
    IN  NDIS_STATUS     Status,
    IN  UINT            BytesTransferred);

VOID
BindToLowerMP(
    OUT PNDIS_STATUS                Status,
    IN  NDIS_HANDLE                 BindContext,
    IN  PNDIS_STRING                MPDeviceName,
    IN  PVOID                       SystemSpecific1,
    IN  PVOID                       SystemSpecific2);

VOID
LowerMPOpenAdapterComplete(
    IN  NDIS_HANDLE ProtocolBindingContext,
    IN  NDIS_STATUS Status,
    IN  NDIS_STATUS OpenErrorStatus);

NDIS_STATUS
MPInitialize(
    OUT PNDIS_STATUS                OpenErrorStatus,
    OUT PUINT                       SelectedMediumIndex,
    IN  PNDIS_MEDIUM                MediumArray,
    IN  UINT                        MediumArraySize,
    IN  NDIS_HANDLE                 MiniportAdapterHandle,
    IN  NDIS_HANDLE                 WrapperConfigurationContext);

PADAPTER
FindAdapterByName(
    PWCHAR AdapterName);

VOID
UnbindFromLowerMP(
    OUT PNDIS_STATUS                Status,
    IN  NDIS_HANDLE                 ProtocolBindingContext,
    IN  NDIS_HANDLE                 UnbindContext);

VOID
DerefAdapter(
    PADAPTER Adapter);

VOID
CleanupAdapter(
    PADAPTER Adapter);

VOID
LowerMPCloseAdapterComplete(
    IN  NDIS_HANDLE ProtocolBindingContext,
    IN  NDIS_STATUS Status);

VOID
CLUnloadProtocol(
    VOID);

VOID
MPHalt(
    IN  NDIS_HANDLE                 MiniportAdapterContext);

NDIS_STATUS
MPReset(
    OUT PBOOLEAN                    AddressingReset,
    IN  NDIS_HANDLE                 MiniportAdapterContext);

```

```

NDIS_STATUS
MPQueryInformation(
     IN  NDIS_HANDLE              MiniportAdapterContext,
     IN  NDIS_OID                 Oid,
     IN  PVOID                    InformationBuffer,
     IN  ULONG                    InformationBufferLength,
     OUT PULONG                   BytesWritten,
     OUT PULONG                   BytesNeeded);

NDIS_STATUS
MPSetInformation(
     IN  NDIS_HANDLE              MiniportAdapterContext,
     IN  NDIS_OID                 Oid,
     IN  PVOID                    InformationBuffer,
     IN  ULONG                    InformationBufferLength,
     OUT PULONG                   BytesRead,
     OUT PULONG                   BytesNeeded);

VOID
CLRequestComplete(
     IN  NDIS_HANDLE      ProtocolBindingContext,
     IN  PNDIS_REQUEST    NdisRequestBuf,
     IN  NDIS_STATUS      Status);

NDIS_STATUS
MakeLocalNdisRequest(
     PADAPTER Adapter,
     NDIS_OID Oid,
     PVOID Buffer,
     ULONG BufferSize);

NDIS_STATUS
MakeLocalNdisRequestSet(
     PADAPTER Adapter,
     NDIS_OID Oid,
     PVOID Buffer,
     ULONG BufferSize);


NTSTATUS
WDMInitialize(
     PDRIVER_OBJECT DriverObject,
     PULONG InitShutdownMask);

VOID
WDMCleanup(
     ULONG ShutdownMask);

NTSTATUS
ConfigureDriver (
     IN PUNICODE_STRING RegistryPath,
     IN PCONFIG_DATA ConfigurationInfo);

VOID
CLStatusIndication(
     IN  NDIS_HANDLE ProtocolBindingContext,
     IN  NDIS_STATUS GeneralStatus,
     IN  PVOID       StatusBuffer,
     IN  UINT        StatusBufferSize);

VOID
CLStatusIndicationComplete(
     IN  NDIS_HANDLE BindingContext);

VOID
CLResetComplete(
     IN  NDIS_HANDLE ProtocolBindingContext,
     IN  NDIS_STATUS Status);


VOID
TNSClientWorkerThread(PVOID Context);

VOID
TNSServerWorkerThread(PVOID Context);

[illegible]
[illegible]
[illegible]

#define      RFCTYPELEN_BEUI       0x80d5
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h** **Page 8 of 11**

```
#define     RFCTYPELEN_IPX       0x8137
#define     RFCTYPELEN_IP        0x800
#define     RFCTYPELEN_ARP       0x806
#define     RFCTYPELEN_APPLE     0x80F3
#define     RFCTYPELEN_XNS       0x600
#define     RFCTYPELEN_RASAUTH   0x8fff

#define     TNS_EMULATION_ETHERTYPE      0xc001 // supposed to be cool
#define     MIN_MTU_PADDING_SIZE         64

//
// These are the TNS client-to-smn and smn-to-client 'command'
// or 'packet-type' indicators.
//
enum {
    TNS_HELLO_BROADCAST=1,
    TNS_HELLO_REPLY,
    TNS_HELLO_GOINGDOWN,        // High priority broadcast packet
    TNS_READ_REQUEST,
    TNS_READ_REPLY,
    TNS_STRING_READ_REQUEST,
    TNS_STRING_READ_REPLY,
    TNS_WRITE_REQUEST,
    TNS_WRITE_ACK,
    TNS_STRING_WRITE_REQUEST,
    TNS_STRING_WRITE_ACK,
    TNS_ACQUIRE_LOCK_REQUEST,
    TNS_RELEASE_LOCK_REQUEST,
    TNS_RELEASE_LOCK_ACK,
    TNS_ALLOCATE_LOCK_REQUEST,
    TNS_ALLOCATE_LOCK_REPLY,
    TNS_DOORBELL_REQUEST,
    TNS_DOORBELL_NOTIFICATION,
    TNS_DOORBELL_NOTIFICATION_ACK,
    TNS_ATOMIC_COMPLEX_ALLOCATE_REQUEST,
    TNS_ATOMIC_COMPLEX_ALLOCATE_REPLY,
    TNS_ATOMIC_COMPLEX_READ_REQUEST,
    TNS_ATOMIC_COMPLEX_READ_REPLY,
    TNS_ATOMIC_COMPLEX_WRITE_REQUEST,
    TNS_ATOMIC_COMPLEX_WRITE_REPLY,
    TNS_INTERLOCKED_ENQUEUE,
    TNS_INTERLOCKED_DEQUEUE,
    TNS_READ_MONOTONIC_COUNTER_REQUEST,
    TNS_READ_MONOTONIC_COUNTER_REPLY,
    TNS_QUERY_STATS,
    TNS_QUERY_STATS_REPLY,
    TNS_QUERY_NODE_INFO,
    TNS_QUERY_NODE_INFO_REPLY,
    TNS_CLEAR_STATS,
};

typedef struct _TNSPacketHeader {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

} TNSPacketHeader, *PTNSPacketHeader;

typedef struct _TNSPacketHelloBroadcast {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    LARGE_INTEGER   RequestStartTSC;
    unsigned char   ClientMacAddress[HARDWARE_ADDRESS_LENGTH];
    unsigned char   ClientMachineName[MAX_COMPUTER_NAME_SIZE];

} TNSPacketHelloBroadcast, *PTNSPacketHelloBroadcast;


typedef struct _TNSPacketHelloReply {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    unsigned char   SMNServerMacAddress[HARDWARE_ADDRESS_LENGTH];
    ULONG           TNSClientNodeID;
    ULONG           TNSSharedMemorySize;
    LARGE_INTEGER   RequestStartTSC;
    ULONG           SMNMachineNameSize;
    unsigned char   SMNMachineName[MAX_COMPUTER_NAME_SIZE];

} TNSPacketHelloReply, *PTNSPacketHelloReply;


typedef struct _TNSPacketReadRequest {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    unsigned long   RequestWidth;
    unsigned long   RequestLength;
    ULONG           RequestOffset;
    LARGE_INTEGER   RequestStartTSC;

} TNSPacketReadRequest, *PTNSPacketReadRequest;


typedef struct _TNSPacketReadReply {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    unsigned long   RequestLength;
    LARGE_INTEGER   RequestStartTSC;
    ULONG           dwData;

} TNSPacketReadReply, *PTNSPacketReadReply;

typedef struct _TNSPacketWriteRequest {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    unsigned long   RequestWidth;
    unsigned long   RequestLength;
    ULONG           RequestOffset;
    ULONG           dwData;
    USHORT          wData;
    UCHAR           bData;
    LARGE_INTEGER RequestStartTSC;

} TNSPacketWriteRequest, *PTNSPacketWriteRequest;


typedef struct _TNSPacketWriteReply {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    unsigned long   RequestWidth;
    unsigned long   RequestLength;
    ULONG           RequestOffset;
    ULONG           dwData;
    USHORT          wData;
    UCHAR           bData;
    LARGE_INTEGER RequestStartTSC;

} TNSPacketWriteReply, *PTNSPacketWriteReply;


typedef struct _TNSPacketQueryStats {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    LARGE_INTEGER   RequestStartTSC;

} TNSPacketQueryStats, *PTNSPacketQueryStats;
```

```

typedef struct _TNSPacketQueryStatsReply {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    LARGE_INTEGER   RequestStartTSC;
    MPSTATS         MpStats;
    NDIS_STATUS     NdisStatus;
    STATISTICS      TnsNodeStatistics;

} TNSPacketQueryStatsReply, *PTNSPacketQueryStatsReply;


typedef struct _TNSPacketQueryNodeInfo {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    LARGE_INTEGER   RequestStartTSC;
    unsigned long   ClientNodeID;

} TNSPacketQueryNodeInfo, *PTNSPacketQueryNodeInfo;

typedef struct _TNSPacketQueryNodeInfoReply {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    LARGE_INTEGER   RequestStartTSC;

    //
    // If node ID comes back 0xffffffff then that node does not exist.
    // Node IDs are assigned sequentially starting at 0, and are always
    // assigned in order.
    //
    unsigned long   ClientNodeID;
    unsigned char   ClientNodeMACAddress[HARDWARE_ADDRESS_LENGTH];
    unsigned char   ClientNodeComputerName[MAX_COMPUTER_NAME_SIZE];

} TNSPacketQueryNodeInfoReply, *PTNSPacketQueryNodeInfoReply;

typedef struct _TNSPacketClearStats {
    unsigned char   MACDstAddress[ETH_ADDRESS_LEN];
    unsigned char   MACSrcAddress[ETH_ADDRESS_LEN];
    unsigned short  MACEtherType;
    unsigned short  TNSCommandReply;

    unsigned long   RequestTag;
    LARGE_INTEGER   RequestStartTSC;
} TNSPacketClearStats, *PTNSPacketClearStats;

#define TNS_PACKET_SIZE(x) (  (sizeof(struct _##x) <= 60) ? 60 : sizeof(struct _##x) )

typedef struct _REQUEST_DATA {
    ULONG           requestOpcode;
    LIST_ENTRY      Linkage;
    unsigned char   TnsPacket[2000];
    PNDIS_PACKET    pNdisPacket;
} REQUEST_DATA, *PREQUEST_DATA;

void
TNSBuildBroadcastReplyAndSend(
    PADAPTER pAdapter,
    PVOID    pTnsPacket,
    unsigned char *pHeader);

unsigned long
TNSGetSharedMemoryNodeNodeID(
    PADAPTER pAdapter,
    unsigned char *pHeader);

VOID
TnsDumpTnsPacket(
    PUCHAR pucBuffer,
    ULONG  bufLength);
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tns.h** **Page 11 of 11**

```

NTSYSAPI
NTSTATUS
NTAPI
ZwAllocateVirtualMemory(
    IN       HANDLE      ProcessHandle,
    IN OUT   PVOID       *BaseAddress,
    IN       ULONG       ZeroBits,
    IN OUT   PULONG      RegionSize,
    IN       ULONG       AllocationType,
    IN       ULONG       Protect);

NTSYSAPI
ULONG
NTAPI
ZwYieldExecution(VOID);

NTSYSAPI
NTSTATUS
NTAPI
ZwFreeVirtualMemory(
    IN       HANDLE      ProcessHandle,
    IN       PVOID       *BaseAddress,
    IN       PULONG      RegionSize,
    IN       ULONG       FreeType);

VOID
TNSSendPackets(
    IN  NDIS_HANDLE              NdisBindingHandle,
    IN  PPNDIS_PACKET            PacketArray,
    IN  UINT                     NumberOfPackets);

NTSTATUS
TNSInitializeClientNodeSendPacket(
    IN       PADAPTER      pAdapter,
    IN OUT   PNDIS_PACKET *ppNdisPacket,
    IN OUT   PVOID        *ppTnsBuffer,
    IN       ULONG         PacketLength);

NDIS_STATUS
TnsGetNICStats(
    PADAPTER    pAdapter,
    pMPSTATS    pMpStats);

int
sprintf(char *s, const char *format, ...);

VOID
TnsIncrementStat(
    PADAPTER pAdapter,
    PLARGE_INTEGER pLi);

VOID
TnsAddStatsUlong(
    PADAPTER pAdapter,
    PLARGE_INTEGER pLi,
    ULONG Added);

void
GetProcessorSpeed(
    PADAPTER pAdapter);

//
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
//
// Status messages, event log messages

//
// MessageId: TNS_EVENT_MINIPORT_REGISTER_FAILED
//
// MessageText:
//
// %2 failed to register as a Intermediate Miniport.
//
#define TNS_EVENT_MINIPORT_REGISTER_FAILED ((NTSTATUS)0xC0080002L)

#endif /* _TNS_H_ */

```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.c** **Page 1 of 8**

```
//**************************************************************************
//
// COPYRIGHT:
//      This program is an unpublished work fully protected by the United
//      States copyright laws and is considered a trade secret belonging to
//      Times N Systems, Inc. To the extent that this work may be
//      considered "published," the following notice applies "© 1999, Times N
//      Systems, Inc." Any unauthorized use, reproduction, distribution,
//      display, modification, or disclosure of this program is strictly
//      prohibited.
//
//**************************************************************************
//++
//**************************************************************************
// Module:
//          tnsdebug.c - Functions to support debug of the emulated subsystem. We
//              (and that includes the mouse in my pocket) support printing
//                  decoded strings for NDIS STATUS, NDIS Events, and OIDs.
//
// Description:
//
// Environment:
//          Windows NT Kernel Mode, Ndis driver models.
//
// Exports:
//            See Module functions generated by script processing.
//
// Author:
//            Vince Bridgers
//            vinceb@timesn.com
//
//--
//**************************************************************************

#include <stdarg.h>
#include <stdio.h>
#include <ndis.h>
#include "tnsdebug.h"
#include "x86.h"

//========================================================================
// Define the proto for the hidden (undocumented, whatever) HAL function
// to make a beep.
//========================================================================

NTHALAPI
BOOLEAN
HalMakeBeep(ULONG Freq);


#ifdef DBG

ULONG _gDebugPrintLevel = 0;                      // flag to control debug output verbosity
ULONG _gDebugPrintMask  = DEBUG_MASKEN_INIT ;     // flag to control debug output verbosity
ULONG _gDebugBreakFlag = TRUE;                    // flag to control if we execute dbg breaks

//**************************************************************************
//++
char *
GetNDISOidString(
    NDIS_OID NdisOID,          // INPUT: NDIS OID to convert to string
    PULONG  pFoundFlag)        // OUTPUT: Flag set to TRUE if found, FALSE if not.
//
// Description:
//     This function returns a ptr to a string type description for the OID parameter.
//
// Environment:
//     Kernel mode only.
//
// Return Value:
//     None.
//
//--
//**************************************************************************
{
    int i;

    typedef struct _NDISOidTable{
        NDIS_OID NdisOID;
        char *OidString;
    } NDISOidTable, *pNDISOidTable;

    static NDISOidTable NDISOidStringTable[] = {
        { OID_802_3_PERMANENT_ADDRESS , "OID_802_3_PERMANENT_ADDRESS", },
        { OID_802_3_CURRENT_ADDRESS   , "OID_802_3_CURRENT_ADDRESS", },
        { OID_802_3_MULTICAST_LIST    , "OID_802_3_MULTICAST_LIST", },
        { OID_802_3_MAXIMUM_LIST_SIZE , "OID_802_3_MAXIMUM_LIST_SIZE", },
        { OID_802_3_MAC_OPTIONS       , "OID_802_3_MAC_OPTIONS", },
        { OID_GEN_SUPPORTED_LIST,   "OID_GEN_SUPPORTED_LIST", },
        { OID_GEN_SUPPORTED_LIST,   "OID_GEN_SUPPORTED_LIST", },
        { OID_GEN_HARDWARE_STATUS,  "OID_GEN_HARDWARE_STATUS", },
        { OID_GEN_MEDIA_SUPPORTED,  "OID_GEN_MEDIA_SUPPORTED", },
        { OID_GEN_MEDIA_IN_USE, "OID_GEN_MEDIA_IN_USE ", },
        { OID_GEN_MAXIMUM_LOOKAHEAD,    "OID_GEN_MAXIMUM_LOOKAHEAD ", },
        { OID_GEN_MAXIMUM_FRAME_SIZE,   "OID_GEN_MAXIMUM_FRAME_SIZE ", },
        { OID_GEN_LINK_SPEED,   "OID_GEN_LINK_SPEED ", },
        { OID_GEN_TRANSMIT_BUFFER_SPACE,    "OID_GEN_TRANSMIT_BUFFER_SPACE ", },
        { OID_GEN_RECEIVE_BUFFER_SPACE, "OID_GEN_RECEIVE_BUFFER_SPACE ", },
        { OID_GEN_TRANSMIT_BLOCK_SIZE,  "OID_GEN_TRANSMIT_BLOCK_SIZE ", },
        { OID_GEN_RECEIVE_BLOCK_SIZE,   "OID_GEN_RECEIVE_BLOCK_SIZE ", },
        { OID_GEN_VENDOR_ID,    "OID_GEN_VENDOR_ID ", },
        { OID_GEN_VENDOR_DESCRIPTION,   "OID_GEN_VENDOR_DESCRIPTION ", },
        { OID_GEN_CURRENT_PACKET_FILTER,    "OID_GEN_CURRENT_PACKET_FILTER ", },
        { OID_GEN_CURRENT_LOOKAHEAD,    "OID_GEN_CURRENT_LOOKAHEAD ", },
        { OID_GEN_DRIVER_VERSION,   "OID_GEN_DRIVER_VERSION ", },
        { OID_GEN_MAXIMUM_TOTAL_SIZE,   "OID_GEN_MAXIMUM_TOTAL_SIZE ", },
        { OID_GEN_PROTOCOL_OPTIONS, "OID_GEN_PROTOCOL_OPTIONS ", },
        { OID_GEN_MAC_OPTIONS,  "OID_GEN_MAC_OPTIONS ", },
        { OID_GEN_MEDIA_CONNECT_STATUS, "OID_GEN_MEDIA_CONNECT_STATUS ", },
        { OID_GEN_MAXIMUM_SEND_PACKETS, "OID_GEN_MAXIMUM_SEND_PACKETS ", },
        { OID_GEN_VENDOR_DRIVER_VERSION,    "OID_GEN_VENDOR_DRIVER_VERSION ", },
        { OID_GEN_XMIT_OK,  "OID_GEN_XMIT_OK ", },
        { OID_GEN_RCV_OK,   "OID_GEN_RCV_OK ", },
        { OID_GEN_XMIT_ERROR,   "OID_GEN_XMIT_ERROR ", },
        { OID_GEN_RCV_ERROR,    "OID_GEN_RCV_ERROR ", },
        { OID_GEN_RCV_NO_BUFFER,    "OID_GEN_RCV_NO_BUFFER ", },
        { OID_GEN_DIRECTED_BYTES_XMIT,  "OID_GEN_DIRECTED_BYTES_XMIT ", },
        { OID_GEN_DIRECTED_FRAMES_XMIT, "OID_GEN_DIRECTED_FRAMES_XMIT ", },
        { OID_GEN_MULTICAST_BYTES_XMIT, "OID_GEN_MULTICAST_BYTES_XMIT ", },
        { OID_GEN_MULTICAST_FRAMES_XMIT,    "OID_GEN_MULTICAST_FRAMES_XMIT ", },
        { OID_GEN_BROADCAST_BYTES_XMIT, "OID_GEN_BROADCAST_BYTES_XMIT ", },
        { OID_GEN_BROADCAST_FRAMES_XMIT,    "OID_GEN_BROADCAST_FRAMES_XMIT ", },
        { OID_GEN_DIRECTED_BYTES_RCV,   "OID_GEN_DIRECTED_BYTES_RCV ", },
        { OID_GEN_DIRECTED_FRAMES_RCV,  "OID_GEN_DIRECTED_FRAMES_RCV ", },
        { OID_GEN_MULTICAST_BYTES_RCV,  "OID_GEN_MULTICAST_BYTES_RCV ", },
        { OID_GEN_MULTICAST_FRAMES_RCV, "OID_GEN_MULTICAST_FRAMES_RCV ", },
        { OID_GEN_BROADCAST_BYTES_RCV,  "OID_GEN_BROADCAST_BYTES_RCV ", },
        { OID_GEN_BROADCAST_FRAMES_RCV, "OID_GEN_BROADCAST_FRAMES_RCV ", },
        { OID_GEN_RCV_CRC_ERROR,    "OID_GEN_RCV_CRC_ERROR ", },
        { OID_GEN_TRANSMIT_QUEUE_LENGTH,    "OID_GEN_TRANSMIT_QUEUE_LENGTH ", },
    };
    #define NUM_NDIS_OID_STRING_ENTRIES (sizeof NDISOidStringTable / sizeof(struct _NDISOidTable))

    #define NDIS_OID_NOT_FOUND_STR   "NDIS OID Code Not Found"

    *pFoundFlag = FALSE;
    for (i=0; i<NUM_NDIS_OID_STRING_ENTRIES; i++) {
        if (NdisOID == NDISOidStringTable[i].NdisOID) {
            *pFoundFlag = TRUE;
            return NDISOidStringTable[i].OidString;
        }
    }
    BreakPoint();
    return NDIS_OID_NOT_FOUND_STR;
}

//***********************************************************************************
//++
char *
GetNDISStatusString(
    NDIS_STATUS Status,         // INPUT:  NDIS Status to convert to string
    PULONG pFoundFlag)          // OUTPUT: flag that says TRUE if found, FALSE if not
//
// Description:
//     From an NDIS status, produce a descriptive string.
//
// Environment:
//     Kernel mode only.
//
// Return Value:
//     None.
//
//--
//***********************************************************************************
```

```
{
    int i;

    //
    // Make structure def and table within scope of this function only
    // not module scope.
    //
    typedef struct _NDISStatusTable{
        NDIS_STATUS Status;
        char *StatusString;
    } NDISStatusTable, *pNDISStatusTable;

    static NDISStatusTable NDISStatusStringTable[] = {
        { NDIS_STATUS_SUCCESS,          "NDIS_STATUS_SUCCESS", },
        { NDIS_STATUS_PENDING,          "NDIS_STATUS_PENDING", },
        { NDIS_STATUS_NOT_RECOGNIZED,   "NDIS_STATUS_NOT_RECOGNIZED", },
        { NDIS_STATUS_NOT_COPIED,       "NDIS_STATUS_NOT_COPIED", },
        { NDIS_STATUS_NOT_ACCEPTED,     "NDIS_STATUS_NOT_ACCEPTED", },
        { NDIS_STATUS_CALL_ACTIVE,      "NDIS_STATUS_CALL_ACTIVE", },
        { NDIS_STATUS_ONLINE,           "NDIS_STATUS_ONLINE", },
        { NDIS_STATUS_RESET_START,      "NDIS_STATUS_RESET_START", },
        { NDIS_STATUS_RESET_END,        "NDIS_STATUS_RESET_END", },
        { NDIS_STATUS_RING_STATUS,      "NDIS_STATUS_RING_STATUS", },
        { NDIS_STATUS_CLOSED,           "NDIS_STATUS_CLOSED", },
        { NDIS_STATUS_WAN_LINE_UP,      "NDIS_STATUS_WAN_LINE_UP", },
        { NDIS_STATUS_WAN_LINE_DOWN,    "NDIS_STATUS_WAN_LINE_DOWN", },
        { NDIS_STATUS_WAN_FRAGMENT,     "NDIS_STATUS_WAN_FRAGMENT", },
        { NDIS_STATUS_MEDIA_CONNECT,    "NDIS_STATUS_MEDIA_CONNECT", },
        { NDIS_STATUS_MEDIA_DISCONNECT, "NDIS_STATUS_MEDIA_DISCONNECT", },
        { NDIS_STATUS_HARDWARE_LINE_UP, "NDIS_STATUS_HARDWARE_LINE_UP", },
        { NDIS_STATUS_HARDWARE_LINE_DOWN, "NDIS_STATUS_HARDWARE_LINE_DOWN", },
        { NDIS_STATUS_INTERFACE_UP,     "NDIS_STATUS_INTERFACE_UP", },
        { NDIS_STATUS_INTERFACE_DOWN,   "NDIS_STATUS_INTERFACE_DOWN", },
        { NDIS_STATUS_MEDIA_BUSY,       "NDIS_STATUS_MEDIA_BUSY", },
        { NDIS_STATUS_WW_INDICATION,    "NDIS_STATUS_WW_INDICATION", },
        { NDIS_STATUS_LINK_SPEED_CHANGE,"NDIS_STATUS_LINK_SPEED_CHANGE", },
        { NDIS_STATUS_NOT_RESETTABLE,   "NDIS_STATUS_NOT_RESETTABLE", },
        { NDIS_STATUS_SOFT_ERRORS,      "NDIS_STATUS_SOFT_ERRORS", },
        { NDIS_STATUS_HARD_ERRORS,      "NDIS_STATUS_HARD_ERRORS", },
        { NDIS_STATUS_BUFFER_OVERFLOW,  "NDIS_STATUS_BUFFER_OVERFLOW", },
        { NDIS_STATUS_FAILURE,          "NDIS_STATUS_FAILURE", },
        { NDIS_STATUS_RESOURCES,        "NDIS_STATUS_RESOURCES", },
        { NDIS_STATUS_CLOSING,          "NDIS_STATUS_CLOSING", },
        { NDIS_STATUS_BAD_VERSION,      "NDIS_STATUS_BAD_VERSION", },
        { NDIS_STATUS_BAD_CHARACTERISTICS, "NDIS_STATUS_BAD_CHARACTERISTICS", },
        { NDIS_STATUS_ADAPTER_NOT_FOUND, "NDIS_STATUS_ADAPTER_NOT_FOUND", },
        { NDIS_STATUS_OPEN_FAILED,      "NDIS_STATUS_OPEN_FAILED", },
        { NDIS_STATUS_DEVICE_FAILED,    "NDIS_STATUS_DEVICE_FAILED", },
        { NDIS_STATUS_MULTICAST_FULL,   "NDIS_STATUS_MULTICAST_FULL", },
        { NDIS_STATUS_MULTICAST_EXISTS, "NDIS_STATUS_MULTICAST_EXISTS", },
        { NDIS_STATUS_MULTICAST_NOT_FOUND, "NDIS_STATUS_MULTICAST_NOT_FOUND", },
        { NDIS_STATUS_REQUEST_ABORTED,  "NDIS_STATUS_REQUEST_ABORTED", },
        { NDIS_STATUS_RESET_IN_PROGRESS, "NDIS_STATUS_RESET_IN_PROGRESS", },
        { NDIS_STATUS_CLOSING_INDICATING, "NDIS_STATUS_CLOSING_INDICATING", },
        { NDIS_STATUS_NOT_SUPPORTED,    "NDIS_STATUS_NOT_SUPPORTED", },
        { NDIS_STATUS_INVALID_PACKET,   "NDIS_STATUS_INVALID_PACKET", },
        { NDIS_STATUS_OPEN_LIST_FULL,   "NDIS_STATUS_OPEN_LIST_FULL", },
        { NDIS_STATUS_ADAPTER_NOT_READY, "NDIS_STATUS_ADAPTER_NOT_READY", },
        { NDIS_STATUS_ADAPTER_NOT_OPEN, "NDIS_STATUS_ADAPTER_NOT_OPEN", },
        { NDIS_STATUS_NOT_INDICATING,   "NDIS_STATUS_NOT_INDICATING", },
        { NDIS_STATUS_INVALID_LENGTH,   "NDIS_STATUS_INVALID_LENGTH", },
        { NDIS_STATUS_INVALID_DATA,     "NDIS_STATUS_INVALID_DATA", },
        { NDIS_STATUS_BUFFER_TOO_SHORT, "NDIS_STATUS_BUFFER_TOO_SHORT", },
        { NDIS_STATUS_INVALID_OID,      "NDIS_STATUS_INVALID_OID", },
        { NDIS_STATUS_ADAPTER_REMOVED,  "NDIS_STATUS_ADAPTER_REMOVED", },
        { NDIS_STATUS_UNSUPPORTED_MEDIA, "NDIS_STATUS_UNSUPPORTED_MEDIA", },
        { NDIS_STATUS_GROUP_ADDRESS_IN_USE, "NDIS_STATUS_GROUP_ADDRESS_IN_USE", },
        { NDIS_STATUS_FILE_NOT_FOUND,   "NDIS_STATUS_FILE_NOT_FOUND", },
        { NDIS_STATUS_ERROR_READING_FILE, "NDIS_STATUS_ERROR_READING_FILE", },
        { NDIS_STATUS_ALREADY_MAPPED,   "NDIS_STATUS_ALREADY_MAPPED", },
        { NDIS_STATUS_RESOURCE_CONFLICT, "NDIS_STATUS_RESOURCE_CONFLICT", },
        { NDIS_STATUS_NO_CABLE,         "NDIS_STATUS_NO_CABLE", },
        { NDIS_STATUS_INVALID_SAP,      "NDIS_STATUS_INVALID_SAP", },
        { NDIS_STATUS_SAP_IN_USE,       "NDIS_STATUS_SAP_IN_USE", },
        { NDIS_STATUS_INVALID_ADDRESS,  "NDIS_STATUS_INVALID_ADDRESS", },
        { NDIS_STATUS_VC_NOT_ACTIVATED, "NDIS_STATUS_VC_NOT_ACTIVATED", },
        { NDIS_STATUS_DEST_OUT_OF_ORDER,"NDIS_STATUS_DEST_OUT_OF_ORDER", },
        { NDIS_STATUS_VC_NOT_AVAILABLE, "NDIS_STATUS_VC_NOT_AVAILABLE", },
        { NDIS_STATUS_CELLRATE_NOT_AVAILABLE, "NDIS_STATUS_CELLRATE_NOT_AVAILABLE", },
        { NDIS_STATUS_INCOMPATABLE_QOS, "NDIS_STATUS_INCOMPATABLE_QOS", },
        { NDIS_STATUS_AAL_PARAMS_UNSUPPORTED, "NDIS_STATUS_AAL_PARAMS_UNSUPPORTED", },
        { NDIS_STATUS_NO_ROUTE_TO_DESTINATION, "NDIS_STATUS_NO_ROUTE_TO_DESTINATION", },
        { NDIS_STATUS_TOKEN_RING_OPEN_ERROR, "NDIS_STATUS_TOKEN_RING_OPEN_ERROR", },
    };

    #define NUM_NDIS_STATUS_STRING_ENTRIES (sizeof NDISStatusStringTable / sizeof(struct _NDISStatusTable
 -2 ))
    #define NDIS_STATUS_NOT_FOUND_STR   "NDIS Status Code Not Found"

    *pFoundFlag = FALSE;
    for (i=0; i<NUM_NDIS_STATUS_STRING_ENTRIES; i++) {
        if (Status == NDISStatusStringTable[i].Status) {
            *pFoundFlag = TRUE;
            return NDISStatusStringTable[i].StatusString;
        }
    }
    BreakPoint();
    return NDIS_STATUS_NOT_FOUND_STR;
}

//****************************************************************************
//-+
char *GetNDISEventString(
    NDIS_ERROR_CODE ErrorCode,              // INPUT:  NDIS error code
    PULONG pFoundFlag)                      // OUTPUT: TRUE if code found, FALSE if not.
//
// Description:
//    Function to take an NDIS ERROR code and produce a string.
//
// Environment:
//    Kernel mode only.
//
// Return Value:
//    None.
//
//-=
//****************************************************************************
{
    int i;

    //
    // Make structure def and table within scope of this function only,
    // not module scope.
    //
    typedef struct _NDISEventTable{
        NDIS_ERROR_CODE ErrorCode;
        char *ErrorCodeString;
    } NDISEventTable, *pNDISEventTable;

    static NDISEventTable NDISEventStringTable[] = {
        { NDIS_ERROR_CODE_RESOURCE_CONFLICT, "NDIS_ERROR_CODE_RESOURCE_CONFLICT", },
        { NDIS_ERROR_CODE_OUT_OF_RESOURCES, "NDIS_ERROR_CODE_OUT_OF_RESOURCES", },
        { NDIS_ERROR_CODE_HARDWARE_FAILURE, "NDIS_ERROR_CODE_HARDWARE_FAILURE", },
        { NDIS_ERROR_CODE_ADAPTER_NOT_FOUND, "NDIS_ERROR_CODE_ADAPTER_NOT_FOUND", },
        { NDIS_ERROR_CODE_INTERRUPT_CONNECT, "NDIS_ERROR_CODE_INTERRUPT_CONNECT", },
        { NDIS_ERROR_CODE_DRIVER_FAILURE, "NDIS_ERROR_CODE_DRIVER_FAILURE", },
        { NDIS_ERROR_CODE_BAD_VERSION, "NDIS_ERROR_CODE_BAD_VERSION", },
        { NDIS_ERROR_CODE_TIMEOUT, "NDIS_ERROR_CODE_TIMEOUT", },
        { NDIS_ERROR_CODE_NETWORK_ADDRESS, "NDIS_ERROR_CODE_NETWORK_ADDRESS", },
        { NDIS_ERROR_CODE_UNSUPPORTED_CONFIGURATION, "NDIS_ERROR_CODE_UNSUPPORTED_CONFIGURATION", },
        { NDIS_ERROR_CODE_INVALID_VALUE_FROM_ADAPTER, "NDIS_ERROR_CODE_INVALID_VALUE_FROM_ADAPTER", },
        { NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER, "NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMET
 -2 ER", },
        { NDIS_ERROR_CODE_BAD_IO_BASE_ADDRESS, "NDIS_ERROR_CODE_BAD_IO_BASE_ADDRESS", },
        { NDIS_ERROR_CODE_RECEIVE_SPACE_SMALL, "NDIS_ERROR_CODE_RECEIVE_SPACE_SMALL", },
        { NDIS_ERROR_CODE_ADAPTER_DISABLED, "NDIS_ERROR_CODE_ADAPTER_DISABLED", },
    };

    #define NUM_NDIS_EVENT_STRING_ENTRIES (sizeof NDISEventStringTable / sizeof(struct _NDISEventTable))
    #define NDIS_EVENT_NOT_FOUND_STR   "NDIS Event Code Not Found"

    *pFoundFlag = FALSE;
    for (i=0; i<NUM_NDIS_EVENT_STRING_ENTRIES; i++) {
        if (ErrorCode == NDISEventStringTable[i].ErrorCode) {
            *pFoundFlag = TRUE;
            return NDISEventStringTable[i].ErrorCodeString;
        }
    }

    return NDIS_EVENT_NOT_FOUND_STR;
}

//****************************************************************************
//-+
VOID
DebugPrint(
    ULONG DebugPrintLevel,           // INPUT: Debug print level
    PCSZ DebugMessage,               // INPUT: Ptr to formatted print string, ala printf
    ...)
//
// Description:
//     Debug print routine.
//
// Environment:
//     Kernel mode only.
//
// Return Value:
//     None.
//
//--
//****************************************************************************
{
    va_list ap;
    va_start(ap, DebugMessage);
    if ( (DebugPrintLevel <= _gDebugPrintLevel) || (DebugPrintLevel == DEBUG_ERROR) ) {
        CHAR buffer[512];

        (VOID) vsprintf(buffer, DebugMessage, ap);

        DbgPrint(buffer);
        if (DebugPrintLevel == DEBUG_ERROR) {
            if (_gDebugBreakFlag) {
                //
                // use an int 3 so we can patch it easier
                //
                // DbgBreakPoint();
                _asm int 3
            }
        }
    }
    va_end(ap);
}


//****************************************************************************
//++
VOID
MaskDebugPrint(
    ULONG DebugPrintLevel,           // INPUT: Debug print level
    ULONG DebugPrintMask,            // INPUT: Debug print mask
    PCSZ DebugMessage,               // INPUT: Ptr to formatted print string, ala printf
    ...)
//
// Description:
//     Debug print routine.
//
// Environment:
//     Kernel mode only.
//
// Return Value:
//     None.
//
//--
//****************************************************************************
{
    va_list ap;
    va_start(ap, DebugMessage);

    if (DebugPrintMask & _gDebugPrintMask) {
        if ( (DebugPrintLevel <= _gDebugPrintLevel) || (DebugPrintLevel == DEBUG_ERROR) ) {
            CHAR buffer[512];

            (VOID) vsprintf(buffer, DebugMessage, ap);

            DbgPrint(buffer);
            if (DebugPrintLevel == DEBUG_ERROR) {
                if (_gDebugBreakFlag) {

                    //
                    // use an int 3 so we can patch it easier
                    //
                    // DbgBreakPoint();
                    _asm int 3
                }
            }
        }
    }

    va_end(ap);
}

//*****************************************************************************
//-+
void
TNSMakeBeep(void)
//
// Description:
//      Performs a 100ms beep at 400Hz, using the undocumented HalMakeBeep
//      function. The way that thing works is to call it with the
//      frequency you want to use for the speaker, wait the desired amount
//      of time, then call it again with a frequency of 0.
//--
//*****************************************************************************
{

    //
    // Start the beep
    //
    HalMakeBeep(400);
    //
    // Stall so the beep is perceptible
    //
    KeStallExecutionProcessor(1000 * 100);
    //
    // Stop the beep by setting the frequency to 0
    //
    HalMakeBeep(0);
}

#define NUMCLOCKSPEEDSAMPLES     100

typedef struct _ProcSpeedData {
    ULONG ProcSpeed;
    ULONG Occurence;
} ProcSpeedData, *pProcSpeedData;


//*****************************************************************************
//-+
VOID
NdisDumpBuffer(
    PUCHAR vaBuffer,                // INPUT: Ptr to contiguous virtual space
    ULONG bufferLength)             // INPUT: Length of space to print
//
// Description:
//      This function dumps the contents of a pool of contiguous virtual memory.
//      For now, we are not dumping the ascii representations.
//
// Environment:
//      Kernel mode only.
//
// Return Value:
//      None.
//
//--
//*****************************************************************************
{
    ULONG i;

    //
    // Disregard the debug print level messages for this function. This function
    // is only called at one place.
    //
    D((0, "%x :", vaBuffer));
    for (i=0; i<bufferLength; i++) {
        if (i%16) {
            D((0, "%02x ", *vaBuffer++));
        } else {
            D((0, "\n%x :", vaBuffer));
            D((0, "%02x ", *vaBuffer++));
        }
    }
    D((0, "\n"));
}

//*****************************************************************************
//-+
VOID
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsdebug.c** **Page 7 of 8**

```
NdisDumpPacket(
    PNDIS_PACKET Packet)        // INPUT: NDIS Packet, what else?
//
// Description:
//    This function dumps the contents of a NDIS packet.
//
// Environment:
//    Kernel mode only.
//
// Return Value:
//    None.
//
//--
//*****************************************************************************
{
    UINT PhysBufferCount, BufferCount, PacketLength;
    PNDIS_BUFFER FirstBuffer, NextBuffer;
    PVOID va;
    UINT bufferLength;
    int i;

    //
    // Get the packet information for this packet and dump it
    //
    NdisQueryPacket(Packet, &PhysBufferCount, &BufferCount, &FirstBuffer, &PacketLength);
    DM((DEBUG_MESSAGE, DEBUG_MASKEN_PACKETDUMP, "DumpPacket: Packet => %x, PhysBufferCount => %d, BufferCount => %d, FirstBuffer => %x, PacketLength => %d\n",
        Packet,
        PhysBufferCount,
        BufferCount,
        FirstBuffer,
        PacketLength));

    //
    // setup our buffers
    //
    NextBuffer = FirstBuffer;

    //
    // walk the buffers, dumping ptr and length information
    //
    for (i=0; NextBuffer!=NULL; i++) {
        NdisQueryBuffer(NextBuffer, &va, &bufferLength);

        DM((DEBUG_MESSAGE, DEBUG_MASKEN_PACKETDUMP, "Buffer => %d, va => %x, bufferLength => %d\n", i, va, bufferLength));

        //
        // Only dump packet contents if we said we want lots of detail
        //
        if ( (_gDebugPrintMask & DEBUG_MASKEN_PACKETDUMP) && (_gDebugPrintLevel >= DEBUG_VERBOSE) ) {
            D((0, "Buffer Contents =>\n"));
            NdisDumpBuffer(va, bufferLength);
        }

        NdisGetNextBuffer(NextBuffer, &NextBuffer);
    }
}


VOID
TnsDumpTnsPacket(
    PUCHAR pucBuffer,
    ULONG  bufLength)
{
    //
    // Dump the destination address
    //
    D((0, "Tns Packet Dest   => %02x-%02x-%02x-%02x-%02x-%02x\n",
        pucBuffer[0],
        pucBuffer[1],
        pucBuffer[2],
        pucBuffer[3],
        pucBuffer[4],
        pucBuffer[5]));

    D((0, "Tns Packet Source => %02x-%02x-%02x-%02x-%02x-%02x\n",
        pucBuffer[6],
        pucBuffer[7],
        pucBuffer[8],
        pucBuffer[9],
        pucBuffer[10],
        pucBuffer[11]));

    D((0, "Tns packet Type   => %02x%02x\n", pucBuffer[12], pucBuffer[13]));
}


#endif //DBG


```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c** **Page 1 of 39**

```
//*************************************************************************
//
//  COPYRIGHT:
//      This program is an unpublished work fully protected by the United
//      States copyright laws and is considered a trade secret belonging to
//      Times N Systems, Inc.  To the extent that this work may be
//      considered "published," the following notice applies: "1999, Times N
//      Systems, Inc."  Any unauthorized use, reproduction, distribution,
//      display, modification, or disclosure of this program is strictly
//      prohibited.
//
//*************************************************************************
//++
//*************************************************************************
// Module:
//        tnsapi.c
//
// Description:
//        This module defines the entry points to emulated Times N Systems
//        services for the multicomputer high-speed interconnect. These
//        calls will be emulated at first, and then later be retargeted to the
//        real hardware.
//
// Environment:
//        Windows NT Kernel Mode only.
//
// Exports:
//        See Module functions generated by script processing.
//
// Author:
//        Vince Bridgers
//        vinceb@timesn.com
//
//--
//*************************************************************************

#include <ntddk.h>
#include <tnsdefs.h>
#include "tns.h"
#include "tnsioctl.h"
#include "tnsdebug.h"
#include "tnsapi.h"
#include "x86.h"


#undef BINARY_COMPATIBLE
#define BINARY_COMPATIBLE 0


NTSTATUS
WDMInitialize(
    PDRIVER_OBJECT DriverObject,
    PULONG InitShutdownMask
    );

VOID
WDMCleanup(
    ULONG ShutdownMask
    );

STATIC NTSTATUS
TNSProcessIOCTLs(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    );


VOID
TNSEmulSetPacketHeader(
    PADAPTER    pAdapter,
    PVOID       pTnsPacket,
    UINT        PacketLength);

unsigned long
TNSGetRequestTag(void);


#pragma NDIS_PAGEABLE_FUNCTION(TNSProcessIOCTLs)

//
// This section defines the functions required for an application to bind
// directly into our driver ioctl function dispatch routine, and to handle
// those calls. In general, we will only export functionality that is
// useful to an application, plus some interesting debug and configuration
// information.
//

NTSTATUS
WDMInitialize(
    PDRIVER_OBJECT DriverObject,
    PULONG InitShutdownMask)
{
    NTSTATUS Status;
    UINT FuncIndex;

    //
    // Initialize the driver object's entry points
    //

    DriverObject->FastIoDispatch = NULL;

    for (FuncIndex = 0; FuncIndex <= IRP_MJ_MAXIMUM_FUNCTION; FuncIndex++) {
        DriverObject->MajorFunction[FuncIndex] = TNSProcessIOCTLs;
    }

    Status = IoCreateDevice(DriverObject,
                            0,
                            &IMDriverName,
                            FILE_DEVICE_NETWORK,
                            0,
                            FALSE,
                            &IMDeviceObject);

    if ( NT_SUCCESS( Status )) {
        *InitShutdownMask |= SHUTDOWN_DELETE_DEVICE;

        IMDeviceObject->Flags |= DO_BUFFERED_IO;

        Status = IoCreateSymbolicLink( &IMSymbolicName, &IMDriverName );

        if ( NT_SUCCESS( Status )) {
            *InitShutdownMask |= SHUTDOWN_DELETE_SYMLINK;
        } else {
            D((0, "IoCreateSymbolic Link Failed (%08X): %ls -> %ls\n", Status, IMSymbolicName.Buffer,
riverName.Buffer));
        }
    } else {
        D((0, "IoCreateDevice Failed - %08x\n", Status ));
        BreakPoint();

        IMDeviceObject = NULL;
    }

    return Status;
}

STATIC NTSTATUS
TNSProcessIOCTLs(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp)
{
    PIO_STACK_LOCATION  irpStack;
    pTNS_IOCTLPACKET    ioBuffer;
    ULONG               inputBufferLength;
    ULONG               outputBufferLength;
    ULONG               ioControlCode;
    NTSTATUS            Status = STATUS_SUCCESS;

    PAGED_CODE();

    //
    // Init to default settings.
    Irp->IoStatus.Status      = STATUS_SUCCESS;
    Irp->IoStatus.Information = 0;

    //
    // Get a pointer to the current location in the Irp. This is where
    //    the function codes and parameters are located.
    //

    irpStack = IoGetCurrentIrpStackLocation(Irp);

    //
    // Get the pointer to the input/output buffer and it's length
    //

    ioBuffer            = (pTNS_IOCTLPACKET)Irp->AssociatedIrp.SystemBuffer;
    inputBufferLength   = irpStack->Parameters.DeviceIoControl.InputBufferLength;
    outputBufferLength  = irpStack->Parameters.DeviceIoControl.OutputBufferLength;

    switch (irpStack->MajorFunction) {
        case IRP_MJ_CREATE:
            D((0, "IRP Create\n"));
            break;

        case IRP_MJ_CLOSE:
            D((0, "IRP Close\n"));
            break;

        case IRP_MJ_CLEANUP:
            D((0, "IRP Cleanup\n"));
            break;

        case IRP_MJ_SHUTDOWN:
            D((0, "IRP Shutdown\n"));
            break;

        case IRP_MJ_DEVICE_CONTROL:

            //
            // get control code from stack and perform the operation
            //

            ioControlCode = irpStack->Parameters.DeviceIoControl.IoControlCode;
            switch (ioControlCode) {

                // This is where you would add your IOCTL handlers
                case IOCTL_TNS_SETDEBUGINFO:
#ifdef DBG
                    _gDebugPrintLevel = ioBuffer->DebugLevel;
                    _gDebugPrintMask  = ioBuffer->DebugMask;
                    _gDebugBreakFlag  = ioBuffer->DebugBreakFlag;
#endif
                    break;

                default:
                    D((0, "unknown IRP_MJ_DEVICE_CONTROL\n = %X\n",ioControlCode));
                    Status = STATUS_INVALID_PARAMETER;
                    BreakPoint();
                    break;

            }
            break;

        default:
            D((0, "unknown IRP major function = %08X\n", irpStack->MajorFunction));
            Status = STATUS_UNSUCCESSFUL;
            BreakPoint();
            break;
    }

    //
    // all requests complete synchronously; notify caller of status
    //

    Irp->IoStatus.Status = Status;
    Irp->IoStatus.Information = outputBufferLength;

    IoCompleteRequest(Irp, IO_NO_INCREMENT);

    return Status;

} // ImIoctl

VOID
WDMCleanup(
    ULONG ShutdownMask)
{
    if ( ShutdownMask & SHUTDOWN_DELETE_SYMLINK ) {
        IoDeleteSymbolicLink( &IMSymbolicName );
    }

    if ( ShutdownMask & SHUTDOWN_DELETE_DEVICE ) {
        IoDeleteDevice( IMDeviceObject );
    }
}
```

```c

void
TNSBuildBroadcastReplyAndSend(
    PADAPTER pAdapter,
    PVOID    pTnsPacket,
    unsigned char *pHeader)
{
    NTSTATUS    Status;
    KIRQL   OldIrql;
    PNDIS_PACKET MyPacket;
    ULONG PacketLength;
    PTNSPacketHelloReply pTnsBuffer;
    PLIST_ENTRY pRequestObj;
    PREQUEST_DATA pRqstData;
    int i;

    //
    // compute packet length, based on request, and
    // set the variable accordingly (the packet structure length
    // will get set according to this variable).
    //
    PacketLength = TNS_PACKET_SIZE(TNSPacketHelloReply);

    Status = TNSInitializeClientNodeSendPacket(pAdapter,
        &MyPacket,
        &pTnsBuffer,
        PacketLength);

    //
    // Set the destination address appropriately
    //
    RtlCopyMemory(pTnsBuffer, &pHeader[6], 6);

    //
    // Fill in relavent packet information here ...
    //
    pTnsBuffer->TNSCommandReply = wswap(TNS_HELLO_REPLY);

    pTnsBuffer->RequestTag = dwswap(((PTNSPacketHelloBroadcast)pTnsPacket)->RequestTag);
    for (i=0; i<HARDWARE_ADDRESS_LENGTH; i++) {
        pTnsBuffer->SMNServerMacAddress[i] = pAdapter->LowerMPMacAddress[i];
    }
    pTnsBuffer->RequestStartTSC = ((PTNSPacketHelloBroadcast)pTnsPacket)->RequestStartTSC;
    pTnsBuffer->TNSClientNodeID = TNSGetSharedMemoryNodeNodeID(pAdapter, pHeader);
    pTnsBuffer->TNSSharedMemorySize = dwswap(pAdapter->TNSSharedMemorySize);

    D((0, "SRV: TNSSharedMemorySize => %x\n", pTnsBuffer->TNSSharedMemorySize));

    //
    // Copy the smn machine name to the reply packet
    //
    for (i=0; i<MAX_COMPUTER_NAME_SIZE; i++) {
        pTnsBuffer->SMNMachineName[i] = pAdapter->LocalComputerName[i];
    }

    //
    // Dequeue a free element from our available object queue
    //
    pRequestObj = ExInterlockedRemoveHeadList(
        &pAdapter->WorkerListEntryPool,
        &pAdapter->ListEntryPoolLock);

    pRqstData = CONTAINING_RECORD(pRequestObj,
                REQUEST_DATA,
                Linkage);

    //
    // tell the server thread what to do
    //

    pRqstData->requestOpcode = TNS_HELLO_REPLY;
    pRqstData->pNdisPacket = MyPacket;

    //
    // Insert object onto server thread object queue
    //
    ExInterlockedInsertTailList(
        &pAdapter->ServerWorkerListEntry,
        &pRqstData->Linkage,
        &pAdapter->ServerWorkerListSpinLock);

    //
    // Now, signal the server thread
    //
    KeReleaseSemaphore(
        &pAdapter->ServerWorkerRequestSemaphore,
        (KPRIORITY) 0,
        (LONG) 1,
        FALSE);

    return;
}

#define MAX_HELLO_RETRIES   20

VOID
TNSClientWorkerThread(
    PVOID Context
    )
{
    NTSTATUS waitStatus;
    LARGE_INTEGER queueWait;
    LARGE_INTEGER waittime;
    PADAPTER serverContext = (PADAPTER)Context;
    PADAPTER pAdapter = (PADAPTER) Context;
    int HelloRetryCount;
    int HelloReceivedReply = FALSE;

    PLIST_ENTRY clientRequest;
    PREQUEST_DATA pClientRequestData;

    ULONG   RegisterData=0xbaddc0de;
    NTSTATUS    Status;
    KIRQL   OldIrql;
    PNDIS_PACKET MyPacket;
    ULONG PacketLength;
    PTNSPacketHelloBroadcast pTnsBuffer;
    int i;

    queueWait.QuadPart = -(3*1000*10000);
    waittime.QuadPart = -(3*10000);

    D((0, "TNSClientWorkerThread\n"));

    KeSetPriorityThread(KeGetCurrentThread(), LOW_REALTIME_PRIORITY+7);

    //
    // build and send our broadcast hello, and wait for a response
    // we need to get the SMN mac address for future
    // transactions
    //

    //
    // Make sure driver has been intialized properly (this is
    // an assertion, this case should never happen).
    //
    //
    // hack hack work on error handling
    //
    while (!pAdapter->TNSDriverInitialized) {
        //
        // wait until the driver has been completely initialized,
        // then continue
        //
        KeDelayExecutionThread(
            KernelMode,
            FALSE,
            &waittime);
    }

    //
    // Raise IRQL to prevent task swapping while we complete processing
    // for this packet.
    //
    //KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);


    if (TNSSharedMemoryNodeEmulation == FALSE) {
        //
        // Compute packet length, based on request, and
        // set the variable accordingly (the packet structure length
        // will get set according to this variable).
        //

        HelloRetryCount = 0;

        while ( (HelloRetryCount++ < MAX_HELLO_RETRIES) && (HelloReceivedReply == FALSE) ) {

            PacketLength = TNS_PACKET_SIZE(TNSPacketHelloBroadcast);
            Status = TNSInitializeClientNodeSendPacket(pAdapter,
                &MyPacket,
                &pTnsBuffer,
                PacketLength);

            D((0, "HelloRetryCount => %d\n", HelloRetryCount));
            //
            // Fill in relavent packet information here...
            //
            pTnsBuffer->TNSCommandReply = wswap(TNS_HELLO_BROADCAST);

            pTnsBuffer->RequestTag = dwswap(TNSGetRequestTag());
            pTnsBuffer->RequestStartTSC = rdtsc();
            for (i=0; i<6; i++) {
                pTnsBuffer->ClientMacAddress[i] = pAdapter->LowerMPMacAddress[i];
            }
            RtlCopyMemory(pTnsBuffer->ClientMachineName, pAdapter->LocalComputerName, MAX_COMPUTER_NAME_SIZE);

            if (NT_SUCCESS(Status)) {
                PLIST_ENTRY wrkrRequest;
                PREQUEST_DATA pWrkrRequestData;
                LARGE_INTEGER queueWait;

                //
                // Send request packet to SMN
                //
                TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);

                //
                // This is a read operation, so we expect a response.
                // Block waiting for the response from the SMN.
                //
                queueWait.QuadPart = -(HelloRetryCount*1000*1000);

                Status = KeWaitForSingleObject(
                    (PVOID) &pAdapter->ClientWorkerResponseSemaphore,
                    Executive,
                    KernelMode,
                    FALSE,
                    &queueWait);

                if (Status == STATUS_TIMEOUT) {
                    //
                    // Do something useful, like inc a stat
                    //

                } else {
                    //
                    // We got a reply.
                    //

                    clientRequest = ExInterlockedRemoveHeadList(
                        &serverContext->ClientWorkerListEntry,
                        &serverContext->ClientWorkerListSpinLock);

                    MyAssert(clientRequest != NULL);

                    pClientRequestData = CONTAINING_RECORD(clientRequest,
                                REQUEST_DATA,
                                Linkage);

                    MyAssert(pClientRequestData != NULL);

                    if (pClientRequestData->requestOpcode != TNS_HELLO_REPLY) {
                        MyAssert(0);
                    } else {
                        D((0, "We got a hello reply\n"));
                        HelloReceivedReply = TRUE;
                    }

                    //
                    // Recycle the queue object
                    //
                    ExInterlockedInsertTailList(&serverContext->WorkerListEntryPool,
                        &pClientRequestData->Linkage,
                        &serverContext->ListEntryPoolLock);
                    }
                }
            }

            while (1) {
                KeDelayExecutionThread(
                    KernelMode,
                    FALSE,
                    &queueWait);

                TnsGetNICStats(pAdapter, &pAdapter->mpStats);
            }
        }



        PsTerminateSystemThread(STATUS_SUCCESS);
}



VOID
TNSServerWorkerThread(
        PVOID Context
        )
{
        NTSTATUS waitStatus;
        LARGE_INTEGER queueWait;
        PADAPTER serverContext = (PADAPTER)Context;
        PADAPTER pAdapter = (PADAPTER)Context;
        PLIST_ENTRY serverRequest;
        PREQUEST_DATA pServerRequestData;
        NTSTATUS Status;

        queueWait.QuadPart = -(3*1000*10000);

        D((0, "TNSServerWorkerThread\n"));

        if (TNSSharedMemoryNodeEmulation) {

            pAdapter->TNSSharedMemoryPtr = NULL;
            pAdapter->TNSSharedMemorySize = 0;

            //_asm int 3
            pAdapter->TNSMemoryType = VIRTUAL_MEMORY;
            pAdapter->TNSMemoryType = NONPAGED_MEMORY;

            if (pAdapter->TNSMemoryType == VIRTUAL_MEMORY) {
                //
                // Make it 4 meg to start with
                //

                pAdapter->TNSSharedMemorySize = 1024*1024*4;

                Status = ZwAllocateVirtualMemory(
                    (HANDLE)     NtCurrentProcess(),
                    (PVOID *)    &pAdapter->TNSSharedMemoryPtr,
                    (ULONG)      0,
                    (PULONG)     &pAdapter->TNSSharedMemorySize,
                    (ULONG)      MEM_COMMIT,
                    (ULONG)      PAGE_READWRITE);

                if (Status != STATUS_SUCCESS) {
                    D((0, "Virtual memory allocation failed\n"));
                    _asm int 3
                } else {
                    D((0, "Virtual memory allocation succeeded\n"));
                    RtlZeroMemory(pAdapter->TNSSharedMemoryPtr, pAdapter->TNSSharedMemorySize);
                }
            }
            if (pAdapter->TNSMemoryType == NONPAGED_MEMORY) {
                //
                // Make it 1 meg to start with
                //
                pAdapter->TNSSharedMemorySize = 1024*1024*1;

                pAdapter->TNSSharedMemoryPtr =
                    ExAllocatePool(
                        NonPagedPool,
                        pAdapter->TNSSharedMemorySize);

            if (pAdapter->TNSSharedMemoryPtr  == NULL) {
                D((0, "NonPagedPool memory allocation failed\n"));
                _asm int 3
            } else {
                D((0, "NonPagedPool  memory allocation succeeded\n"));
                RtlZeroMemory(pAdapter->TNSSharedMemoryPtr, pAdapter->TNSSharedMemorySize);
            }

        }

    }
    KeSetPriorityThread(KeGetCurrentThread(), LOW_REALTIME_PRIORITY+7);

    do {
        waitStatus = KeWaitForSingleObject(
            (PVOID) &serverContext->ServerWorkerRequestSemaphore,
            Executive,
            KernelMode,
            FALSE,
            &queueWait);


        //
        // Check for timeout, if we do, then do something
        //
        if (waitStatus == STATUS_TIMEOUT) {
            //
            // If Status is timeout, take the opportunity to do something useful,
            // and collect some statistical data
            //
            TnsGetNICStats(pAdapter, &pAdapter->mpStats);

            continue;
        }

        //D((0, "TNSServerWorkerThread - dequeued an object\n"));
        MyAssert(waitStatus == STATUS_SUCCESS);

        while (serverRequest = ExInterlockedRemoveHeadList(
            &serverContext->ServerWorkerListEntry,
            &serverContext->ServerWorkerListSpinLock)) {

            pServerRequestData = CONTAINING_RECORD(serverRequest,
                        REQUEST_DATA,
                        Linkage);

            MyAssert(pServerRequestData);

            switch (pServerRequestData->requestOpcode) {
                case TNS_READ_REQUEST: {
                    PNDIS_PACKET MyPacket;
                    ULONG PacketLength;
                    PTNSPacketReadReply pTnsBuffer;
                    NTSTATUS Status;
                    PUCHAR   vBuffer;

                    vBuffer = pAdapter->TNSSharedMemoryPtr;

                    //D((0, "Processing server read request\n"));
                    PacketLength = TNS_PACKET_SIZE(TNSPacketReadReply);

                    Status = TNSInitializeClientNodeSendPacket(pAdapter,
                        &MyPacket,
                        &pTnsBuffer,
                        PacketLength);

                    RtlCopyMemory(pTnsBuffer, &((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->MACSrcAddress, 6);
                    //
                    // Fill in relavent packet information here...
                    //
                    pTnsBuffer->TNSCommandReply = wswap(TNS_READ_REPLY);

                    pTnsBuffer->RequestTag = ((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->RequestTag;
                    pTnsBuffer->RequestStartTSC = ((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->RequestStartTSC;
                    vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap(((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->RequestOffset));

                    if (dwswap( ((PTNSPacketReadRequest)(pServerRequestData->TnsPacket))->RequestOffset) <= pAdapter->TNSSharedMemorySize ) {
                        pTnsBuffer->dwData = *((PULONG)vBuffer);
                    } else {
                        _asm int 3
                    }

                    TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);

                    break;
                    }
                case TNS_WRITE_REQUEST: {
                    PNDIS_PACKET MyPacket;
                    ULONG PacketLength;
                    NTSTATUS Status;
                    PUCHAR   vBuffer;

                    //D((0, "Processing server write request\n"));

                    vBuffer = pAdapter->TNSSharedMemoryPtr;

                    vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap( ((PTNSPacketWriteRequest)(pServerRequestData->TnsPacket))->RequestOffset));

                    if (dwswap(((PTNSPacketWriteRequest)(pServerRequestData->TnsPacket))->RequestOffset) <= pAdapter->TNSSharedMemorySize ) {
                        *((PULONG)vBuffer) = ((PTNSPacketWriteRequest)(pServerRequestData->TnsPacket))->dwData;
                    } else {
                        _asm int 3
                    }

                    break;
                    }
                case TNS_HELLO_REPLY:
                    MyAssert(TNSSharedMemoryNodeEmulation);
                    //
                    // Send hello reply
                    //
                    D((0, "Processing server hello reply\n"));

                    TNSSendPackets(pAdapter->LowerMPHandle, &pServerRequestData->pNdisPacket, 1);

                    break;
                default:
                    MyAssert(0);
                    break;
            }
            //
            // Recycle the queue object
            //
            ExInterlockedInsertTailList(&serverContext->WorkerListEntryPool,
                &pServerRequestData->Linkage,
                &serverContext->ListEntryPoolLock);
        }
    } while (TRUE);

    PsTerminateSystemThread(STATUS_SUCCESS);
}

VOID
TNSEmulSetPacketHeader(
    PADAPTER    pAdapter,
    PVOID       pTnsPacket,
    UINT        PacketLength)
{
    UINT i;
    ULONG *pulData;

    pulData = (PULONG) pTnsPacket;

    //
    // Zero memory (take this out later)
    //
    RtlZeroMemory(pTnsPacket, PacketLength);

    //
    // Put a recognizable pattern into packet buffer
    //
    for (i=0; i<PacketLength/4; i++) {
        *pulData++ = 0xcafebabe;
    }

    //
    // Set the destination and source addresses for the packet
    //
    for (i=0; i<HARDWARE_ADDRESS_LENGTH; i++) {
        ((PTNSPacketHeader)pTnsPacket)->MACDstAddress[i] = pAdapter->SMNMacAddress[i];
        ((PTNSPacketHeader)pTnsPacket)->MACSrcAddress[i] = pAdapter->LowerMPMacAddress[i];
    }
    //
    // Set the ethertype to our ethertype
    //
    ((PTNSPacketHeader)pTnsPacket)->MACEtherType = wswap(TNS_EMULATION_ETHERTYPE);

}

//
// Initialized to 0; incremented by 1 each time we use it. We use
// this to help up keep track of outstanding requests to the SMN.
//
unsigned long _gRequestTag = 0;
unsigned long
TNSGetRequestTag(void)
{
    return _gRequestTag++;
}


//
// Initialized to 0, incremented by 1 each time we use it. We use
// this to help up keep track of outstanding requests to the SMN.
//
unsigned long _gSharedMemoryNodeNodeID = 0;
unsigned long
TNSGetSharedMemoryNodeNodeID(
    PADAPTER pAdapter,
    unsigned char *pHeader)
{
    ULONG i;
    ULONG NextFreeSpace=0xffffffff;
    ULONG NewTeamNodeID;
    PTNSPacketHelloBroadcast pTnsPacket = (PTNSPacketHelloBroadcast) pHeader;

    for (i=0; i<MAX_TEAM_NODES; i++) {
        if (pAdapter->TeamNodeTable[i].LocationSet) {
            if ( RtlCompareMemory(&pHeader[6], pAdapter->TeamNodeTable[i].TNMacAddress, 6) == 6) {
                return pAdapter->TeamNodeTable[i].TNNodeID;
            }
        } else {
            if (NextFreeSpace == 0xffffffff) {
                NextFreeSpace = i;
            }
        }
    }

    //
    // If we made it this far, we did not find an entry.
    // Set an entry in our table for this mac address.
    //
    NewTeamNodeID = _gSharedMemoryNodeNodeID++;
    RtlCopyMemory(pAdapter->TeamNodeTable[NextFreeSpace].TNMacAddress, &pHeader[6], 6);
    RtlCopyMemory(pAdapter->TeamNodeTable[NextFreeSpace].TNComputerName, pTnsPacket->ClientMachineName, M
AX_COMPUTER_NAME_SIZE);
    pAdapter->TeamNodeTable[NextFreeSpace].LocationSet = TRUE;
    pAdapter->TeamNodeTable[NextFreeSpace].TNNodeID = NewTeamNodeID;

    return NewTeamNodeID;
}


LARGE_INTEGER diffTime;

NTSTATUS
TNSInitializeClientNodeSendPacket(
    IN      PADAPTER      pAdapter,
    IN OUT  PNDIS_PACKET  *ppNdisPacket,
    IN OUT  PVOID         *ppTnsBuffer,
    IN      ULONG         PacketLength)
{
    NTSTATUS Status=STATUS_SUCCESS;
    PTNS_PACKET_CONTEXT     PktContext;
    PNDIS_PACKET    MyPacket;
    PNDIS_BUFFER    MyNdisBuffer;
    PVOID      vBuffer;
    NDIS_PHYSICAL_ADDRESS HighAddress = NDIS_PHYSICAL_ADDRESS_CONST( -1, -1 );
    PVOID pTnsPacket;
    LARGE_INTEGER startTime, endTime;

    //
    // Allocate a packet from our available packet pool,
    // check status, reinit the packet, and get the
    // context context area
    //
    startTime = rdtsc();
    NdisAllocatePacket(&Status, &MyPacket, pAdapter->PacketPoolHandle);
    endTime = rdtsc();

    diffTime.QuadPart = endTime.QuadPart - startTime.QuadPart;

    if (diffTime.LowPart > 0x400) {
        //D((0, "NdisAllocatePacket Time => %x\n", diffTime.LowPart));
    }

    //
    // hack hack work on error handling
    //
    if (Status != STATUS_SUCCESS) {
        _asm int 3
        return Status;
    }
    NdisReinitializePacket(MyPacket);

    PktContext = PACKET_CONTEXT_FROM_PACKET(MyPacket);

    PktContext->OriginalPacket = NULL;
    PktContext->LookaheadBuffer = NULL;
    PktContext->SMNEmulationPacket = TRUE;

    //
    // Now, allocate a buffer to chain to the packet
    //
    Status = NdisAllocateMemory(&vBuffer, PacketLength, 0, HighAddress);

    //
    // hack hack work on error handling
    //
    if (Status != NDIS_STATUS_SUCCESS) {
        NdisFreePacket(MyPacket);
        return Status;
    }

    NdisAllocateBuffer(&Status,
        &MyNdisBuffer,
        pAdapter->LookaheadPoolHandle,
        vBuffer,
        PacketLength);

    //
    // hack hack work on error handling
    //
    if (Status != NDIS_STATUS_SUCCESS) {
        _asm int 3
        NdisFreePacket(MyPacket);
        NdisFreeMemory(vBuffer, PacketLength, 0);
        return Status;
    }

    pTnsPacket = (PTNSPacketHelloBroadcast) vBuffer;

    //
    // Setup the packet mac dest, source, and ethertype
    //

    TNSEmulSetPacketHeader(pAdapter, pTnsPacket, PacketLength);

    //
    // Set the packet length
    //
    NdisAdjustBufferLength(MyNdisBuffer, PacketLength);

    //
    // Chain our buffer to this packet structure
    //
    NdisChainBufferAtFront(MyPacket, MyNdisBuffer);
    NdisRecalculatePacketCounts(MyPacket);

    *ppNdisPacket = MyPacket;
    *ppTnsBuffer = pTnsPacket;

    return Status;
}

VOID
TNSFlushReadReplyQueue(
    PADAPTER pAdapter)
{
    LARGE_INTEGER queueWait;
    NTSTATUS Status;
    PLIST_ENTRY clientRequest;
    PREQUEST_DATA pClientRequestData;

    do {
        queueWait.QuadPart = -(0);

        Status = KeWaitForSingleObject(
            (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
            Executive,
            KernelMode,
            FALSE,
            &queueWait);

        if (Status == STATUS_SUCCESS) {

            clientRequest = ExInterlockedRemoveHeadList(
                &pAdapter->ClientWorkerListEntry,
                &pAdapter->ClientWorkerListSpinLock);

            MyAssert(clientRequest != NULL);

            pClientRequestData = CONTAINING_RECORD(clientRequest,
                        REQUEST_DATA,
                        Linkage);

            MyAssert(pClientRequestData);

            TnsIncrementStat(pAdapter, &pAdapter->MyStats.numDiscardedTnsRecvs);

            //
            // Recycle the queue object
            //
            ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
                &pClientRequestData->Linkage,
                &pAdapter->ListEntryPoolLock);
        }
    } while (Status == STATUS_SUCCESS) ;
}



//
// Start Kernel Mode DLL entry points
//

#define MAX_REQUEST_RESPONSE_RETRIES     50

//*****************************************************************************
//
ULONG
DECLSPEC_EXPORT
__TNS_READ_REGISTER_ULONG(
    IN  PVOID   DeviceHandle,
    IN  PULONG  Register)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//
//*****************************************************************************
{
    ULONG   RegisterData=0xbaddc0de;
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    NTSTATUS    Status;
    KIRQL   OldIrql;
    PNDIS_PACKET MyPacket;
    ULONG PacketLength;
    PTNSPacketReadRequest pTnsBuffer;
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c** **Page 13 of 3**

```
PLIST_ENTRY clientRequest;
PREQUEST_DATA pClientRequestData;
ULONG requestTag;
ULONG retries=0;
int noreply = TRUE;
ULONG returnRequestTag;
LARGE_INTEGER startTime, endTime, diffTime;

//
// hack hack. We really wanna use the device context given up
// by the caller.
//
pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

//
// Raise IRQL to prevent task swapping while we complete processing
// for this packet.
//
KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);

//
// Make sure driver has been intialized properly (this is
// an assertion, this case should never happen).
//
//
// hack hack work on error handling
//
if (!pAdapter->TNSDriverInitialized) {
    BreakPoint();
    KeLowerIrql(OldIrql);
    return 0;
}

TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadRequests);
//
// compute packet length, based on request, and
// set the variable accordingly (the packet structure length
// will get set according to this variable).
//

PacketLength = TNS_PACKET_SIZE(TNSPacketReadRequest);

requestTag = TNSGetRequestTag();

while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {

    Status = TNSInitializeClientNodeSendPacket(pAdapter,
        &MyPacket,
        &pTnsBuffer,
        PacketLength);

    //
    // Fill in relavent packet information here...
    //
    pTnsBuffer->TNSCommandReply = wswap(TNS_READ_REQUEST);

    pTnsBuffer->RequestTag = dwswap(requestTag);
    pTnsBuffer->RequestWidth = dwswap(4);
    pTnsBuffer->RequestLength = dwswap(1);
    pTnsBuffer->RequestOffset = dwswap((unsigned long)Register);
    pTnsBuffer->RequestStartTSC = rdtsc();

    if (NT_SUCCESS(Status)) {
        PLIST_ENTRY wrkrRequest;
        PREQUEST_DATA pWrkrRequestData;
        LARGE_INTEGER queueWait;
        int timeout = FALSE;
        int ltimeout = FALSE;
        int timeoutcount = 0;

        //
        // Flush the read reply queue. In case a different request timed out,
        // and it actually shows up, we need to flush the queue for
        // subsequent requests.
        //
        TNSFlushReadReplyQueue(pAdapter);

        startTime = rdtsc();
        //
        // Send request packet to SMN
        //
        TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
```

```

            //
            // This is a read operation, so we expect a response.
            // Block waiting for the response from the SMN.
            //
            // this is 100m secs.
            //

            queueWait.QuadPart = -(1000000);

            Status = KeWaitForSingleObject(
                (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
                Executive,
                KernelMode,
                FALSE,
                &queueWait);

            if (Status != STATUS_TIMEOUT) {
                PTNSPacketReadReply pTnsPacketReadReply;

                clientRequest = ExInterlockedRemoveHeadList(
                    &pAdapter->ClientWorkerListEntry,
                    &pAdapter->ClientWorkerListSpinLock);

                MyAssert(clientRequest != NULL);

                pClientRequestData = CONTAINING_RECORD(clientRequest,
                            REQUEST_DATA,
                            Linkage);

                MyAssert(pClientRequestData != NULL);
                pTnsPacketReadReply = (PTNSPacketReadReply) &pClientRequestData->TnsPacket;

                RegisterData      =  pTnsPacketReadReply->dwData;
                returnRequestTag = dwswap(pTnsPacketReadReply->RequestTag);

                //MyAssert(returnRequestTag == requestTag);

                if (returnRequestTag == requestTag) {
                    noreply = FALSE;
                    endTime = rdtsc();
                }

                //
                // only maintain stats if we did not retry the operation
                //
                if ( (retries == 1) && (noreply==FALSE) ) {
                    diffTime.QuadPart = endTime.QuadPart - startTime.QuadPart;
                    if (pAdapter->MyStats.maxReadTimeSingle.QuadPart == 0) {
                        pAdapter->MyStats.maxReadTimeSingle.QuadPart = diffTime.QuadPart;
                    } else {
                        if (diffTime.QuadPart > pAdapter->MyStats.maxReadTimeSingle.QuadPart) {
                            pAdapter->MyStats.maxReadTimeSingle.QuadPart = diffTime.QuadPart;
                        }
                    }
                    if (pAdapter->MyStats.minReadTimeSingle.QuadPart == 0) {
                        pAdapter->MyStats.minReadTimeSingle.QuadPart = diffTime.QuadPart;
                    } else {
                        if (diffTime.QuadPart < pAdapter->MyStats.minReadTimeSingle.QuadPart) {
                            pAdapter->MyStats.minReadTimeSingle.QuadPart = diffTime.QuadPart;
                        }
                    }
                    if (pAdapter->MyStats.numReadTimeSingleSamples.QuadPart < 30000) {
                        pAdapter->MyStats.cumReadTimeSingle.QuadPart += diffTime.QuadPart;
                        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadTimeSingleSamples);
                    } else {
                        pAdapter->MyStats.cumReadTimeSingle.QuadPart = diffTime.QuadPart;
                        pAdapter->MyStats.numReadTimeSingleSamples.QuadPart = 1;
                    }
                }

                //
                // Recycle the queue object
                //
                ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
                    &pClientRequestData->Linkage,
                    &pAdapter->ListEntryPoolLock);
            } else {
                TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadRequestTimeouts);
            }
        }
    }

    KeLowerIrql(OldIrql);

    if (retries > 1) {
        TnsAddStatsUlong(pAdapter, &pAdapter->MyStats.numWriteRequestRetries, retries-1);
    }

    if (noreply == TRUE) {
        RegisterData = 0xFFFFFFFF;

        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numReadRequestNoReplies);
        //
        // Throw an exception to our client ....
        //
        // TODO
    }

    return RegisterData;
}



//********************************************************************************
//++
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_ULONG(
    IN  PVOID   DeviceHandle,
    IN  PULONG  Register,
    IN  ULONG   RegisterData)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//********************************************************************************
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    NTSTATUS    Status;
    KIRQL   OldIrql;
    PNDIS_PACKET MyPacket;
    ULONG PacketLength;
    PTNSPacketWriteRequest pTnsBuffer;
    ULONG requestTag;
    ULONG retries=0;
    int noreply = TRUE;
    PLIST_ENTRY clientRequest;
    PREQUEST_DATA pClientRequestData;
    ULONG returnRequestTag;
    LARGE_INTEGER startTime, endTime, diffTime;


    //D((0, "TNS_WRITE_REGISTER_ULONG\n"));

    //
    // hack hack: We really wanna use the device context given up
    // by the caller.
    //
    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    //
    // Raise IRQL to prevent task swapping while we complete processing
    // for this packet.
    //
    KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);

    //
    // Make sure driver has been intialized properly (this is
    // an assertion, this case should never happen).
    //
    //
    // hack hack work on error handling
    //
    if (!pAdapter->TNSDriverInitialized) {
        BreakPoint();
        KeLowerIrql(OldIrql);
        return;
    }

```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c** **Page 1 of 3**

```
TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteRequests);

//
// compute packet length, based on request, and
// set the variable accordingly (the packet structure length
// will get set according to this variable).
//

PacketLength = TNS_PACKET_SIZE(TNSPacketWriteRequest);
requestTag = TNSGetRequestTag();

while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {

    Status = TNSInitializeClientNodeSendPacket(pAdapter,
        &MyPacket,
        &pTnsBuffer,
        PacketLength);

    //
    // Fill in relavent packet information here...
    //
    pTnsBuffer->TNSCommandReply = wswap(TNS_WRITE_REQUEST);

    pTnsBuffer->RequestTag = dwswap(requestTag);
    pTnsBuffer->RequestWidth = dwswap(4);
    pTnsBuffer->RequestLength = dwswap(1);
    pTnsBuffer->RequestOffset = dwswap((unsigned long)Register);
    pTnsBuffer->dwData = RegisterData;
    pTnsBuffer->RequestStartTSC = rdtsc();

    if (NT_SUCCESS(Status)) {
        PLIST_ENTRY wrkrRequest;
        PREQUEST_DATA pWrkrRequestData;
        LARGE_INTEGER queueWait;

        TNSFlushReadReplyQueue(pAdapter);

        startTime = rdtsc();
        //
        // Send request packet to SMN (We assume realiable delivery)
        //
        TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);

        queueWait.QuadPart = -(1000000);

        Status = KeWaitForSingleObject(
            (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
            Executive,
            KernelMode,
            FALSE,
            &queueWait);

        if (Status != STATUS_TIMEOUT) {
            PTNSPacketWriteReply pTnsWriteReplyPacket;

            clientRequest = ExInterlockedRemoveHeadList(
                &pAdapter->ClientWorkerListEntry,
                &pAdapter->ClientWorkerListSpinLock);

            MyAssert(clientRequest != NULL);

            pClientRequestData = CONTAINING_RECORD(clientRequest,
                        REQUEST_DATA,
                        Linkage);

            MyAssert(pClientRequestData != NULL);

            pTnsWriteReplyPacket = (PTNSPacketWriteReply)&pClientRequestData->TnsPacket;

            returnRequestTag = dwswap(pTnsWriteReplyPacket->RequestTag);

            //MyAssert(returnRequestTag == requestTag);

            if (returnRequestTag == requestTag) {
                noreply = FALSE;
                endTime = rdtsc();
            }

            if ( (retries == 1) && (noreply==FALSE) ) {
                diffTime.QuadPart = endTime.QuadPart - startTime.QuadPart;
                if (pAdapter->MyStats.maxWriteTimeSingle.QuadPart == 0) {
                    pAdapter->MyStats.maxWriteTimeSingle.QuadPart = diffTime.QuadPart;
```

```
                        } else {
                            if (diffTime.QuadPart > pAdapter->MyStats.maxWriteTimeSingle.QuadPart) {
                                pAdapter->MyStats.maxWriteTimeSingle.QuadPart = diffTime.QuadPart;
                            }
                        }
                        if (pAdapter->MyStats.minWriteTimeSingle.QuadPart == 0) {
                            pAdapter->MyStats.minWriteTimeSingle.QuadPart = diffTime.QuadPart;
                        } else {
                            if (diffTime.QuadPart < pAdapter->MyStats.minWriteTimeSingle.QuadPart) {
                                pAdapter->MyStats.minWriteTimeSingle.QuadPart = diffTime.QuadPart;
                            }
                        }

                        if (pAdapter->MyStats.numWriteTimeSingleSamples.QuadPart < 30000) {
                            pAdapter->MyStats.cumWriteTimeSingle.QuadPart += diffTime.QuadPart;
                            TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteTimeSingleSamples);
                        } else {
                            pAdapter->MyStats.cumWriteTimeSingle.QuadPart = diffTime.QuadPart;
                            pAdapter->MyStats.numWriteTimeSingleSamples.QuadPart = 1;
                        }
                    }

                    //
                    // Recycle the queue object
                    //
                    ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
                        &pClientRequestData->Linkage,
                        &pAdapter->ListEntryPoolLock);

                } else {

                    TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteRequestTimeouts);

                }
            }
        }

        if (retries > 1) {
            TnsAddStatsUlong(pAdapter, &pAdapter->MyStats.numWriteRequestRetries, retries-1);
        }


        if (noreply == TRUE) {
            //
            // Throw an exception to our client - TODO
            //


            TnsIncrementStat(pAdapter, &pAdapter->MyStats.numWriteRequestNoReplies);
        }

        KeLowerIrql(OldIrql);

        return;
    }

    //*****************************************************************************
    //-+
    USHORT
    DECLSPEC_EXPORT
    __TNS_READ_REGISTER_USHORT(
        IN  PVOID   DeviceHandle,
        IN  PUSHORT Register)
    //
    // Description:
    //
    // Environment:
    //
    // Return Value:
    //
    //--
    //*****************************************************************************
    {
        USHORT RegisterData=0xbadd;

        return RegisterData;
    }

    //*****************************************************************************
    //-+
    VOID
    DECLSPEC_EXPORT
```

```
__TNS_WRITE_REGISTER_USHORT(
    IN  PVOID   DeviceHandle,
    IN  PUSHORT Register,
    IN  USHORT  RegisterData)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*****************************************************************************
{
}

//*****************************************************************************
//++
UCHAR
DECLSPEC_EXPORT
__TNS_READ_REGISTER_UCHAR(
    IN  PVOID   DeviceHandle,
    IN  PUCHAR  Register)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*****************************************************************************
{
    UCHAR  RegisterData=0xba;

    return RegisterData;
}

//*****************************************************************************
//++
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_UCHAR(
    IN  PVOID   DeviceHandle,
    IN  PUCHAR  Register,
    IN  UCHAR   RegisterData)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*****************************************************************************
{
}



//*****************************************************************************
//++
VOID
DECLSPEC_EXPORT
__TNS_READ_REGISTER_BUFFER_ULONG(
    IN  PVOID   DeviceHandle,
    IN  PULONG  Register,
    IN  PULONG  pulBuffer,
    IN  ULONG   Count)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*****************************************************************************
{
}

//*****************************************************************************
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c** **Page 19 of 39**

```
//++
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_BUFFER_ULONG(
    IN  PVOID   DeviceHandle,
    IN  PULONG  Register,
    IN  PULONG  pulBuffer,
    IN  ULONG   Count)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//****************************************************************************
{
}

//****************************************************************************
//++
VOID
DECLSPEC_EXPORT
__TNS_READ_REGISTER_BUFFER_USHORT(
    IN  PVOID   DeviceHandle,
    IN  PUSHORT Register,
    IN  PUSHORT pusBuffer,
    IN  ULONG   Count)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//****************************************************************************
{
}

//****************************************************************************
//++
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_BUFFER_USHORT(
    IN  PVOID   DeviceHandle,
    IN  PUSHORT Register,
    IN  PUSHORT pusBuffer,
    IN  ULONG   Count)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//****************************************************************************
{
}




//****************************************************************************
//++
VOID
DECLSPEC_EXPORT
__TNS_READ_REGISTER_BUFFER_UCHAR(
    IN  PVOID   DeviceHandle,
    IN  PUCHAR  Register,
    IN  PUCHAR  pucBuffer,
    IN  ULONG   Count)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
```

```
//*****************************************************************************
{
}

//*****************************************************************************
//+
VOID
DECLSPEC_EXPORT
__TNS_WRITE_REGISTER_BUFFER_UCHAR(
     IN  PVOID   DeviceHandle,
     IN  PUCHAR  Register,
     IN  PUCHAR  pucBuffer,
     IN  ULONG   Count)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
}

//*****************************************************************************
//+
TNS_STATUS
DECLSPEC_EXPORT
__TNSAcquireLockP(
     IN  PVOID   DeviceHandle,
     IN  PLOCKID pLockID)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
     return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//+
TNS_STATUS
DECLSPEC_EXPORT
__TNSReleaseLockP(
     IN  PVOID   DeviceHandle,
     IN  PLOCKID pLockID)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
     return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//+
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueryLockP(
     IN  PVOID       DeviceHandle,
     OUT PLOCKSTATUS pLockStatus)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}



//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSAllocateLockP(
    IN  PVOID   DeviceHandle,
    IN  TNSKEY  Key,
    OUT PLOCKID *pLockID)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSFreeLockP(
    IN  PVOID   DeviceHandle,
    IN  TNSKEY  Key,
    IN  PLOCKID pLockID)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSNotifyCPU(
    IN  PVOID       DeviceHandle,
    IN  TNSCPUID    CpuID,
    IN  PVOID       pMessageBuffer,
    IN  ULONG       MessageLength)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSNotifyCPUSync(
    IN  PVOID       DeviceHandle,
    IN  TNSCPUID    CpuID,
    IN  PVOID       pMessageBuffer,
    IN  ULONG       MessageLength,
    IN  PVOID       pCallback,
    IN  PVOID       pContext)
//
// Description:
```

```
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueryNotifyStatus(
    IN      PVOID               DeviceHandle,
    IN      TNSCPUID            CpuID,
    IN  OUT PTNSNOTIFYSTATUS    pCpuNotifyInfo)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSRegisterNotifyCallback(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pCallBack,
    IN  PVOID       SysParm1,
    IN  PVOID       SysParm2,
    IN  PVOID       SysParm3)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSRegisterNotificationCallback(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pCallBack,
    IN  PVOID       SysParm1,
    IN  PVOID       SysParm2,
    IN  PVOID       SysParm3)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


```

```
//***************************************************************************
//-+
TNS_STATUS
DECLSPEC_EXPORT
__TNSDeRegisterNotificationCallback(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pCallBack)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//***************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//***************************************************************************
//-+
TNSCPUID
DECLSPEC_EXPORT
__TNSWhoAmI(
    IN  PVOID       DeviceHandle)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//***************************************************************************
{
    return 0;
}

//***************************************************************************
//-+
TNSCOUNTER
DECLSPEC_EXPORT
__TNSReadOrdinalCounter(
    IN  PVOID       DeviceHandle)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//***************************************************************************
{
    return 0;
}


//***************************************************************************
//-+
TNS_STATUS
DECLSPEC_EXPORT
__TNSAllocateSharedMemory(
    IN      PVOID         DeviceHandle,
    IN      TNSKEY        Key,
    IN      TNSMEMFLAGS   Flags,
    IN      TNSMEMSIZE    Size,
    IN OUT  PVOID         *ppBuffer)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//***************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}
```

```

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSFreeSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  TNSKEY      Key,
    IN  PVOID       Ptr,
    IN  TNSMEMSIZE  Size)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSReadSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSWriteSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSDmaReadSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer,
    IN  PVOID       pCallback,
    IN  PVOID       DMAReadCompleteComtext1,
    IN  PVOID       DMAReadCompleteComtext2)
//
// Description:
```

```
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSDmaWriteSharedMemory(
    IN  PVOID       DeviceHandle,
    IN  PVOID       pSharedMemoryAddress,
    IN  ULONG       Length,
    IN  PVOID       pBuffer,
    IN  PVOID       pCallback,
    IN  PVOID       DMAWriteCompleteComtext1,
    IN  PVOID       DMAWriteCompleteComtext2)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSAllocateWorkQueue(
    IN      PVOID       DeviceHandle,
    IN      TNSKEY      Key,
    IN      PULONG      pQueueLength,
    IN OUT  PTNSQUEUE   *ppTNSQueue)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSFreeWorkQueue(
    IN      PVOID       DeviceHandle,
    IN      TNSKEY      Key,
    IN      PTNSQUEUE   pTNSQueue)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedEnqueueToDoP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN      PVOID       pItem,
    IN      ULONG       Length)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedDequeueToDoP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN      PVOID       pItem,
    IN      PULONG      pLength)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueryQLengthP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN      PULONG      pLength)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueueHeadP(
    IN      PVOID       DeviceHandle,
    IN      PTNSQUEUE   pTNSQueue,
    IN OUT  PTNSQUEUE   *ppTNSQueue)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
```

```
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueueTailP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN OUT  PTNSQUEUE    *ppTNSQueue)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueuePayloadP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN      PVOID        pItem,
    IN      PULONG       pLength)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueueNextP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN OUT  PTNSQUEUE    *ppTNSQueue)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedInsertQueueItemP(
    IN      PVOID        DeviceHandle,
    IN      PTNSQUEUE    pTNSQueue,
    IN      PTNSQUEUE    pTNSQueueInsert)
//
// Description:
//
```

```
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSInterlockedDeleteQueueItemP(
    IN      PVOID         DeviceHandle,
    IN      PTNSQUEUE     pTNSQueue,
    IN      PTNSQUEUE     pTNSQueueDelete)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSQueueItemInfoP(
    IN      PVOID            DeviceHandle,
    IN      PTNSQUEUE        pTNSQueue,
    IN      PTNSQUEUEINFO    pTNSQueueInfo)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}


//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSGetFirstDeviceInstance(
    PVOID   *ppDeviceInstance)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//*****************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}

//*****************************************************************************
//++
TNS_STATUS
DECLSPEC_EXPORT
__TNSGetNextDeviceInstance(
    PVOID   pDeviceInstance,
    PVOID   *ppDeviceInstance)
//
// Description:
```

```
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*************************************************************************
{
    return TNS_STATUS_NOT_IMPLEMENTED;
}



//*************************************************************************
//++
ULONG
DECLSPEC_EXPORT
__TNS_GET_SMN_STATISTICS(
    IN      PVOID       DeviceHandle,
    IN OUT  PSTATISTICS pStatistics,
    IN OUT  PULONG      pStatsStructSize,
    IN OUT  pMPSTATS    pMpStats,
    IN OUT  PULONG      pMpStatsSize)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*************************************************************************
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    NTSTATUS    Status;
    KIRQL   OldIrql;
    PNDIS_PACKET MyPacket;
    ULONG PacketLength;
    PTNSPacketQueryStats pTnsBuffer;
    PLIST_ENTRY clientRequest;
    PREQUEST_DATA pClientRequestData;
    ULONG requestTag;
    ULONG retries=0;
    int noreply = TRUE;
    ULONG returnRequestTag;

    //
    // hack hack: We really wanna use the device context given up
    // by the caller.
    //
    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    //
    // Raise IRQL to prevent task swapping while we complete processing
    // for this packet.
    //
    KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);

    //
    // Make sure driver has been intialized properly (this is
    // an assertion, this case should never happen).
    //
    //
    // hack hack work on error handling
    //
    if (!pAdapter->TNSDriverInitialized) {
        BreakPoint();
        KeLowerIrql(OldIrql);
        return 0;
    }

    //
    // compute packet length, based on request, and
    // set the variable accordingly (the packet structure length
    // will get set according to this variable).
    //

    PacketLength = TNS_PACKET_SIZE(TNSPacketQueryStats);

    requestTag = TNSGetRequestTag();

    while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {
```

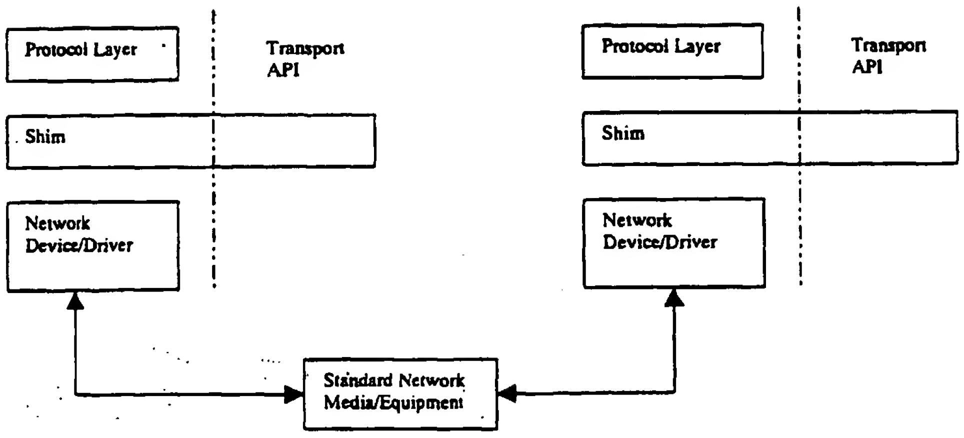
Protocol Layer
Transport API
Shim
Network Device/Driver
Standard Network Media/Equipment
Protocol Layer
Transport API
Shim
Network Device/Driver

FIGURE 1

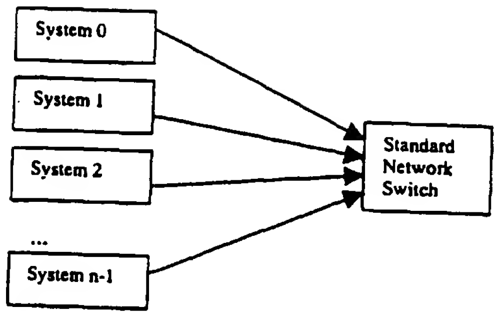

FIGURE 2

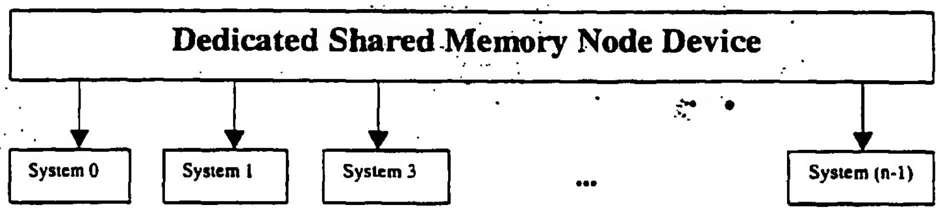

FIGURE 3

```

        Status = TNSInitializeClientNodeSendPacket(pAdapter,
            &MyPacket,
            &pTnsBuffer,
            PacketLength);

        //
        // Fill in relavent packet information here....
        //
        pTnsBuffer->TNSCommandReply = wswap(TNS_QUERY_STATS);

        pTnsBuffer->RequestTag = dwswap(requestTag);
        pTnsBuffer->RequestStartTSC = rdtsc();

        if (NT_SUCCESS(Status)) {
            PLIST_ENTRY wrkrRequest;
            PREQUEST_DATA pWrkrRequestData;
            LARGE_INTEGER queueWait;
            int timeout = FALSE;
            int ltimeout = FALSE;
            int timeoutcount = 0;

            //
            // Flush the read reply queue. In case a different request timed out,
            // and it actually shows up, we need to flush the queue for
            // subsequent requests.
            //
            TNSFlushReadReplyQueue(pAdapter);

            //
            // Send request packet to SMN
            //
            TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);

            //
            // This is a read operation, so we expect a response.
            // Block waiting for the response from the SMN.
            //
            // this is 100m secs.
            //
            queueWait.QuadPart = -(1000000);

            Status = KeWaitForSingleObject(
                (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
                Executive,
                KernelMode,
                FALSE,
                &queueWait);

            if (Status != STATUS_TIMEOUT) {
                PTNSPacketQueryStatsReply pTnsPacketQueryStatsReply;

                clientRequest = ExInterlockedRemoveHeadList(
                    &pAdapter->ClientWorkerListEntry,
                    &pAdapter->ClientWorkerListSpinLock);

                MyAssert(clientRequest != NULL);

                pClientRequestData = CONTAINING_RECORD(clientRequest,
                            REQUEST_DATA,
                            Linkage);

                MyAssert(pClientRequestData != NULL);

                pTnsPacketQueryStatsReply = (PTNSPacketQueryStatsReply) &pClientRequestData->TnsPacke

                returnRequestTag    = dwswap(pTnsPacketQueryStatsReply->RequestTag);
                MyAssert(returnRequestTag == requestTag);

                if (returnRequestTag == requestTag) {
                    noreply = FALSE;
                    RtlCopyMemory(pStatistics, &pTnsPacketQueryStatsReply->TnsNodeStatistics, sizeof(
 ISTICS) );
                    RtlCopyMemory(pMpStats, &pTnsPacketQueryStatsReply->MpStats, sizeof(MPSTATS) );
                }
                //
                // Recycle the queue object
                //
                ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
                    &pClientRequestData->Linkage,
                    &pAdapter->ListEntryPoolLock);
            } else {
                    //
                    // do something useful ?
                    //
                }
            }
        }

        KeLowerIrql(OldIrql);

        if (noreply == TRUE) {
            //
            // Throw an exception to our client ...
            //
            // TODO
        }

        return 0;
}


//*****************************************************************************
//-+
ULONG
DECLSPEC_EXPORT
__TNS_GET_SMN_STATISTICS_BY_NODEID(
        IN      PVOID       DeviceHandle,
        IN      ULONG       NodeID,
        IN OUT  PSTATISTICS pStatistics,
        IN OUT  PULONG      pStatsStructSize,
        IN OUT  pMPSTATS    pMpStats,
        IN OUT  PULONG      pMpStatsSize)
//
// Description:
//
// Environment:
//
// Return Value:
//
//-
//*****************************************************************************
{
        PADAPTER pAdapter = (PADAPTER) DeviceHandle;
        NTSTATUS    Status;
        KIRQL   OldIrql;
        PNDIS_PACKET MyPacket;
        ULONG PacketLength;
        PTNSPacketQueryStats pTnsBuffer;
        PLIST_ENTRY clientRequest;
        PREQUEST_DATA pClientRequestData;
        ULONG requestTag;
        ULONG retries=0;
        int noreply = TRUE;
        ULONG returnRequestTag;
        ULONG retValue = 0;

        //
        // hack hack. We really wanna use the device context given up
        // by the caller.
        //
        pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

        if (TNSSharedMemoryNodeEmulation) {

            //
            // Find index into SMN node info table, make sure
            // it's valid.
            //
            if (NodeID < MAX_TEAM_NODES) {
                if (pAdapter->TeamNodeTable[NodeID].LocationSet == 0) {
                    return 0;
                }
            } else {
                return 0;
            }

            //
            // Raise IRQL to prevent task swapping while we complete processing
            // for this packet.
            //
            KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);

```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsapi.c** **Page 32 of 39**

```
        //
        // Make sure driver has been intialized properly (this is
        // an assertion, this case should never happen).
        //
        //
        // hack hack work on error handling
        //
        if (!pAdapter->TNSDriverInitialized) {
            BreakPoint();
            KeLowerIrql(OldIrql);
            return 0;
        }

        //
        // compute packet length, based on request, and
        // set the variable accordingly (the packet structure length
        // will get set according to this variable).
        //

        PacketLength = TNS_PACKET_SIZE(TNSPacketQueryStats);

        requestTag = TNSGetRequestTag();

        while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {

            Status = TNSInitializeClientNodeSendPacket(pAdapter,
                &MyPacket,
                &pTnsBuffer,
                PacketLength);

            //
            // Set directed packet address by node id
            //
            RtlCopyMemory(
                pTnsBuffer->MACDstAddress,
                pAdapter->TeamNodeTable[NodeID].TNMacAddress,
                ETH_ADDRESS_LEN);

            //
            // Fill in relavent packet information here...
            //
            pTnsBuffer->TNSCommandReply = wswap(TNS_QUERY_STATS);

            pTnsBuffer->RequestTag = dwswap(requestTag);
            pTnsBuffer->RequestStartTSC = rdtsc();

            if (NT_SUCCESS(Status)) {
                PLIST_ENTRY wrkrRequest;
                PREQUEST_DATA pWrkrRequestData;
                LARGE_INTEGER queueWait;
                int timeout = FALSE;
                int ltimeout = FALSE;
                int timeoutcount = 0;

                //
                // Flush the read reply queue. In case a different request timed out,
                // and it actually shows up, we need to flush the queue for
                // subsequent requests.
                //
                TNSFlushReadReplyQueue(pAdapter);

                //
                // Send request packet to SMN
                //
                TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);

                //
                // This is a read operation, so we expect a response.
                // Block waiting for the response from the SMN.
                //
                // this is 100m-secs.
                //
                queueWait.QuadPart = -(1000000);

                Status = KeWaitForSingleObject(
                    (PVOID) &pAdapter->ClientWorkerRequestSemaphore,
                    Executive,
                    KernelMode,
                    FALSE,
                    &queueWait);

                if (Status != STATUS_TIMEOUT) {
                        PTNSPacketQueryStatsReply pTnsPacketQueryStatsReply;

                        clientRequest = ExInterlockedRemoveHeadList(
                            &pAdapter->ClientWorkerListEntry,
                            &pAdapter->ClientWorkerListSpinLock);

                        MyAssert(clientRequest != NULL);

                        pClientRequestData = CONTAINING_RECORD(clientRequest,
                                    REQUEST_DATA,
                                    Linkage);

                        MyAssert(pClientRequestData != NULL);

                        pTnsPacketQueryStatsReply = (PTNSPacketQueryStatsReply) &pClientRequestData->TnsPacket;

                        returnRequestTag    = dwswap(pTnsPacketQueryStatsReply->RequestTag);
                        //MyAssert(returnRequestTag == requestTag);

                        if (returnRequestTag == requestTag) {
                            noreply = FALSE;
                            RtlCopyMemory(pStatistics, &pTnsPacketQueryStatsReply->TnsNodeStatistics, sizeof(STATISTICS) );
                            RtlCopyMemory(pMpStats, &pTnsPacketQueryStatsReply->MpStats, sizeof(MPSTATS) );
                            retValue = 1;
                        }
                        //
                        // Recycle the queue object
                        //
                        ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
                            &pClientRequestData->Linkage,
                            &pAdapter->ListEntryPoolLock);
                    } else {
                        //
                        // do something useful ?
                        //
                    }
                }
            }

        KeLowerIrql(OldIrql);

        if (noreply == TRUE) {
            //
            // Throw an exception to our client ...
            //
            // TODO
        }
    } else {
    }

    return 0;
}

//****************************************************************************************
//-+
ULONG
DECLSPEC_EXPORT
__TNS_GET_SMN_INFORMATION(
    IN      PVOID       DeviceHandle,
    IN OUT  unsigned char *pMacAddress,
    IN OUT  unsigned char *pNodeName,
    IN OUT  unsigned long *pSharedMemorySize)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//****************************************************************************************
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    RtlCopyMemory(pMacAddress, &pAdapter->SMNMacAddress, HARDWARE_ADDRESS_LENGTH);
    RtlCopyMemory(pNodeName, &pAdapter->SMNMachineName, 16);
    *pSharedMemorySize = pAdapter->TNSSharedMemorySize;
    return 0;
}
```

```

//****************************************************************************
//--
ULONG
DECLSPEC_EXPORT
__TNS_GET_NODE_INFORMATION(
    IN      PVOID       DeviceHandle,
    IN OUT  unsigned char *pMacAddress,
    IN OUT  unsigned char *pNodeName,
    IN OUT  unsigned int  *pNodeID)
//
// Description:
//
// Environment:
//
// Return Value:
//  [illegible]
//--
//****************************************************************************
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    RtlCopyMemory(pMacAddress, &pAdapter->LowerMPMacAddress, HARDWARE_ADDRESS_LENGTH);
    RtlCopyMemory(pNodeName, &pAdapter->LocalComputerName, 16);
    *pNodeID = pAdapter->TNSClientNodeID;
    return 0;
}

//****************************************************************************
//--
ULONG
DECLSPEC_EXPORT
__TNS_CLEAR_NODE_STATISTICS(
    IN      PVOID       DeviceHandle)
//
// Description:
//
// Environment:
//
// Return Value:
//  [illegible]
//--
//****************************************************************************
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    RtlZeroMemory(&pAdapter->MyStats, sizeof(STATISTICS));
    RtlZeroMemory(&pAdapter->mpStats, sizeof(MPSTATS));
    GetProcessorSpeed(pAdapter);
    return 0;
}



//****************************************************************************
//--
ULONG
DECLSPEC_EXPORT
__TNS_GET_SMN_TABLE_INFO(
    IN      PVOID       DeviceHandle,
    IN OUT  pSMNTableInfo pSMNInfo)
//
// Description:
//
// Environment:
//
// Return Value:
//  [illegible]
//--
//****************************************************************************
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    ULONG retValue=0;
    int i,j;

    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    if (TNSSharedMemoryNodeEmulation) {
        //
        // Return true if we are an SMN
        //
        retValue = 1;
        for (i=0; i<MAX_TEAM_NODES; i++) {
            pSMNInfo->LocationSet = pAdapter->TeamNodeTable[i].LocationSet;
            for (j=0; j<6; j++) {
                pSMNInfo->MacAddress[j] = pAdapter->TeamNodeTable[i].TNMacAddress[j];
            }
            for (j=0; j<MAX_COMPUTER_NAME_SIZE; j++) {
                pSMNInfo->ComputerName[j] = pAdapter->TeamNodeTable[i].TNComputerName[j];
            }
            pSMNInfo->NodeID = pAdapter->TeamNodeTable[i].TNNodeID;
            pSMNInfo++;
        }
    }

    return retValue;
}

//*********************************************************************
//++
ULONG
DECLSPEC_EXPORT
__TNS_CLEAR_SMN_STATISTICS(
    IN       PVOID        DeviceHandle)
//
// Description:
//
// Environment:
//
// Return Value:
// [illegible]
//--
//*********************************************************************
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    NTSTATUS    Status;
    KIRQL   OldIrql;
    PNDIS_PACKET MyPacket;
    ULONG PacketLength;
    PTNSPacketClearStats pTnsBuffer;
    PLIST_ENTRY clientRequest;
    PREQUEST_DATA pClientRequestData;
    ULONG requestTag;
    ULONG retries=0;
    int noreply = TRUE;
    ULONG returnRequestTag;

    //
    // hack hack: we really wanna use the device context given up
    // by the caller.
    //
    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    //
    // Raise IRQL to prevent task swapping while we complete processing
    // for this packet.
    //
    KeRaiseIrql(DISPATCH_LEVEL, &OldIrql);

    //
    // Make sure driver has been intialized properly (this is
    // an assertion, this case should never happen).
    //
    //
    // hack hack work on error handling
    //
    if (!pAdapter->TNSDriverInitialized) {
        BreakPoint();
        KeLowerIrql(OldIrql);
        return 0;
    }

    //
    // compute packet length, based on request, and
    // set the variable accordingly (the packet structure length
    // will get set according to this variable).
    //

    PacketLength = TNS_PACKET_SIZE(TNSPacketClearStats);

    requestTag = TNSGetRequestTag();

```

**File: D:\nt4DDK\sr \timesn\tnsdrvr\tnsapi.c** **Page 36 of 39**

```
    while (noreply && (retries++ < MAX_REQUEST_RESPONSE_RETRIES) ) {

        Status = TNSInitializeClientNodeSendPacket(pAdapter,
            &MyPacket,
            &pTnsBuffer,
            PacketLength);

        //
        // Fill in relavent packet information here...
        //
        pTnsBuffer->TNSCommandReply = wswap(TNS_CLEAR_STATS);

        pTnsBuffer->RequestTag = dwswap(requestTag);
        pTnsBuffer->RequestStartTSC = rdtsc();

        if (NT_SUCCESS(Status)) {
            PLIST_ENTRY wrkrRequest;
            PREQUEST_DATA pWrkrRequestData;
            LARGE_INTEGER queueWait;
            int timeout = FALSE;
            int ltimeout = FALSE;
            int timeoutcount = 0;

            //
            // Send request packet to SMN
            //
            TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
        }
    }
    KeLowerIrql(OldIrql);

    return 0;
}



//[illegible]
//++
ULONG
DECLSPEC_EXPORT
__TNS_GET_NODE_STATISTICS(
    IN      PVOID       DeviceHandle,
    IN OUT  PSTATISTICS pStatistics,
    IN OUT  PULONG      pStatsStructSize,
    IN OUT  pMPSTATS    pMpStats,
    IN OUT  PULONG      pMpStatsSize)
//
// Description:
//
// Environment:
//
// Return Value:
//
//--
//[illegible]
{
    PADAPTER pAdapter = (PADAPTER) DeviceHandle;
    NDIS_STATUS NdisStatus;
    //
    // Hack Hack. We really wanna use the device context given up
    // by the caller.
    //
    pAdapter = CONTAINING_RECORD(AdapterList.Flink, ADAPTER, Linkage);

    MyAssert(pStatsStructSize);
    MyAssert(pMpStatsSize);

    if ( (*pStatsStructSize >= sizeof (STATISTICS)) && (pStatistics) ) {
        RtlCopyMemory(pStatistics, &pAdapter->MyStats, sizeof(STATISTICS) );
    } else {
        *pStatsStructSize = sizeof (STATISTICS);
        return 0;
    }
    if( (*pMpStatsSize >= sizeof (MPSTATS)) && (pMpStats) ) {
        TnsGetNICStats(pAdapter, pMpStats);
    } else {
        *pMpStatsSize = sizeof (MPSTATS) ;
        return 0;
    }

    return 1;
}
```

```




unsigned char zerobuffer[6] = { 0, 0, 0, 0, 0, 0 };

VOID
TNSSendPackets(
    IN  NDIS_HANDLE             NdisBindingHandle,
    IN  PPNDIS_PACKET           PacketArray,
    IN  UINT                    NumberOfPackets)
{
    UINT PhysBufferCount, BufferCount, PacketLength;
    PNDIS_BUFFER FirstBuffer, NextBuffer;
    PUCHAR va;
    UINT bufferLength;
    unsigned short *pEtherType;
    unsigned int i,j;
    NDIS_STATUS Status;
    int Found;


    for (i=0; i<NumberOfPackets; i++) {

#ifdef DBG
        NdisQueryPacket(PacketArray[i], &PhysBufferCount, &BufferCount, &FirstBuffer, &PacketLength);

        NextBuffer = FirstBuffer;
        for (j=0; NextBuffer!= NULL; j++) {
            NdisQueryBuffer(NextBuffer, &va, &bufferLength);

            if (j==0) {
                MyAssert(bufferLength != 0);
                if (bufferLength >= 14) {
                    pEtherType = (unsigned short *)&va[12];
                    MyAssert (wswap(*pEtherType) == TNS_EMULATION_ETHERTYPE);
                    MyAssert (RtlCompareMemory(va, zerobuffer, 6) != 6);
                    MyAssert (RtlCompareMemory(&va[6], zerobuffer, 6) != 6);
                }
            }
            NdisGetNextBuffer(NextBuffer, &NextBuffer);
        }
#endif
        NdisSend(&Status, NdisBindingHandle, PacketArray[i]);

#ifdef DBG
        switch (Status) {
            case NDIS_STATUS_SUCCESS:
                break;
            case NDIS_STATUS_PENDING:
                break;
            case NDIS_STATUS_INVALID_PACKET:
                MyAssert(0);
                break;
            case NDIS_STATUS_CLOSING:
                MyAssert(0);
                break;
            case NDIS_STATUS_RESET_IN_PROGRESS:
                MyAssert(0);
                break;
            case NDIS_STATUS_FAILURE:
                MyAssert(0);
                break;
            default:
                MyAssert(0);
                D((0, "Status => %x, %s\n", Status, GetNDISStatusString(Status, &Found) ));
                break;
        }
#endif

    }
    //NdisSendPackets(NdisBindingHandle, PacketArray, NumberOfPackets);
}

NDIS_STATUS
TnsGetNICStats(
    PADAPTER    pAdapter,
    pMPSTATS    pMpStats)
{
    NDIS_STATUS NdisStatus;

    NdisStatus = MakeLocalNdisRequest(
        pAdapter,
        OID_GEN_XMIT_OK,
        &pMpStats->XmitOK,
        sizeof(ULONG));
    if (NdisStatus != NDIS_STATUS_SUCCESS) {
        //return NdisStatus;
        _asm int 3
    }

    NdisStatus = MakeLocalNdisRequest(
        pAdapter,
        OID_GEN_RCV_OK,
        &pMpStats->RcvOK,
        sizeof(ULONG));
    if (NdisStatus != NDIS_STATUS_SUCCESS) {
        _asm int 3
        //return NdisStatus;
    }

    NdisStatus = MakeLocalNdisRequest(
        pAdapter,
        OID_GEN_XMIT_ERROR,
        &pMpStats->XmitError,
        sizeof(ULONG));
    if (NdisStatus != NDIS_STATUS_SUCCESS) {
        _asm int 3
        //return NdisStatus;
    }

    NdisStatus = MakeLocalNdisRequest(
        pAdapter,
        OID_GEN_RCV_ERROR,
        &pMpStats->RcvError,
        sizeof(ULONG));
    if (NdisStatus != NDIS_STATUS_SUCCESS) {
        _asm int 3
        //return NdisStatus;
    }

    NdisStatus = MakeLocalNdisRequest(
        pAdapter,
        OID_GEN_RCV_NO_BUFFER,
        &pMpStats->RcvNoBuffer,
        sizeof(ULONG));
    if (NdisStatus != NDIS_STATUS_SUCCESS) {
        _asm int 3
        //return NdisStatus;
    }

    NdisStatus = MakeLocalNdisRequest(
        pAdapter,
        OID_GEN_RCV_CRC_ERROR,
        &pMpStats->RcvCrcError,
        sizeof(ULONG));
    if (NdisStatus != NDIS_STATUS_SUCCESS) {
        _asm int 3
        //return NdisStatus;
    }


    return NDIS_STATUS_SUCCESS;
}


VOID
TnsAddStatsUlong(
    PADAPTER pAdapter,
    PLARGE_INTEGER pLi,
    ULONG Addend)
{
    LARGE_INTEGER AddendPart;

    AddendPart.HighPart = 0;
    AddendPart.LowPart = Addend;

    (void)ExInterlockedAddLargeInteger(pLi, AddendPart, &pAdapter->MyStatsLock);
}

VOID
TnsIncrementStat(
    PADAPTER pAdapter,
    PLARGE_INTEGER pLi)
```

```
{
    LARGE_INTEGER Addend;

    Addend.QuadPart = 1;

    (void)ExInterlockedAddLargeInteger(pLi, Addend, &pAdapter->MyStatsLock);
}

unsigned long _fltused;

void
GetProcessorSpeed(
    PADAPTER pAdapter)
{
    LARGE_INTEGER qPerfCounter1, qPerfCounter2, qPerfDiff, qPerfFreq;
    //double freq;

    LARGE_INTEGER qPerfInc = {65536, 0};
    LARGE_INTEGER qrdtsc1, qrdtsc2, qrdtscdiff;

    qPerfCounter1 = KeQueryPerformanceCounter(&qPerfFreq);

    qPerfCounter2.QuadPart = qPerfCounter1.QuadPart + qPerfInc.QuadPart;

    qrdtsc1 = rdtsc();
    do {
        qPerfCounter1 = KeQueryPerformanceCounter(NULL);
        qrdtsc2 = rdtsc();
    } while (qPerfCounter1.QuadPart < qPerfCounter2.QuadPart) ;

    qPerfDiff.QuadPart = qPerfCounter1.QuadPart - (qPerfCounter2.QuadPart - qPerfInc.QuadPart);
    qrdtscdiff.QuadPart = qrdtsc2.QuadPart - qrdtsc1.QuadPart;

    //freq = ((double)qrdtscdiff.LowPart*(double)qPerfFreq.LowPart)/(double)qPerfDiff.LowPart;

    pAdapter->MyStats.rdtscDiff = qrdtscdiff.LowPart;
    pAdapter->MyStats.perfFreq  = qPerfFreq.LowPart;
    pAdapter->MyStats.perfDiff  = qPerfDiff.LowPart;

    D((0, "qrdtscdiff.LowPart => %x\n", qrdtscdiff.LowPart   ));
    D((0, "qPerfFreq.LowPart  -> %x\n", qPerfFreq.LowPart    ));
    D((0, "qPerfDiff.LowPart  => %x\n", qPerfDiff.LowPart    ));
}


```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c** **Page 1 of 20**

```
//*****************************************************************************
//
// COPYRIGHT:
//      This program is an unpublished work fully protected by the United
//      States copyright laws and is considered a trade secret belonging to
//      Times N Systems, Inc.  To the extent that this work may be
//      considered "published," the following notice applies: "1999, Times N
//      Systems, Inc."  Any unauthorized use, reproduction, distribution,
//      display, modification, or disclosure of this program is strictly
//      prohibited.
//
//*****************************************************************************
//++
//*****************************************************************************
// Module:
//      tnsemul.c - Main initialization and support routine module
//      for Times N High Speed Interconnect emulation driver.
//
// Description:
//
// Environment:
//      Windows NT Kernel Mode, Ndis driver models only.
//
// Exports:
//      See Module functions generated by script processing.
//
// Author:
//      Vince Bridgers
//      vinceb@timesn.com
//--
//*****************************************************************************

#include "tns.h"
#include "tnsdebug.h"

PADAPTER CurrentAdapter;
ULONG   TNSSharedMemoryNodeEmulation = FALSE;

NDIS_PHYSICAL_ADDRESS HighAddress = NDIS_PHYSICAL_ADDRESS_CONST( -1, -1 );

LIST_ENTRY AdapterList;
NDIS_SPIN_LOCK AdapterListLock;

NDIS_HANDLE ClientProtocolHandle;

NDIS_HANDLE MPWrapperHandle;

NDIS_HANDLE LMDriverHandle;

PDRIVER_OBJECT IMDriverObject;
PDEVICE_OBJECT IMDeviceObject;

CONFIG_DATA ConfigData;

NDIS_STRING IMSymbolicName = NDIS_STRING_CONST("\\DosDevices\\Im");
NDIS_STRING IMDriverName =   NDIS_STRING_CONST("\\Device\\Im" );
NDIS_STRING IMMPName =       NDIS_STRING_CONST("\\Device\\Im" );

DECLARE_STRING( PacketPoolSize );
DECLARE_STRING( DebugLevel );
DECLARE_STRING( DebugMask );
DECLARE_STRING( TNSSMNEmulationMode );

//*****************************************************************************
//
// Function prototypes
//
//*****************************************************************************

NTSTATUS
DriverEntry(
    IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath);

STATIC NDIS_STATUS
GetAdapterRegistryData(
    PNDIS_STRING IMParamsKey,
    PADAPTER pAdapter);

STATIC VOID
ProcessLowerMPOpenAdapter(
    IN  PADAPTER pAdapter,
    IN  NDIS_STATUS Status);

STATIC NDIS_STATUS
AllocatePacketPool(
  .  PADAPTER pAdapter);

STATIC NDIS_STATUS
AllocateReceiveBufferPools(
     PADAPTER pAdapter);

STATIC ULONG
ReadSingleParameter(
     IN NDIS_HANDLE ParametersHandle,
     IN PWCHAR ValueName,
     IN ULONG DefaultValue,
     IN NDIS_PARAMETER_TYPE ParamType);

STATIC VOID
WriteSingleParameter(
     IN NDIS_HANDLE ParametersHandle,
     IN PWCHAR ValueName,
     IN ULONG ValueData,
     IN NDIS_PARAMETER_TYPE ParamType);

[illegible]
//
//Pragmas [illegible]
//
[illegible]

#ifdef ALLOC_PRAGMA
#pragma alloc_text(INIT, ConfigureDriver)
#pragma alloc_text(INIT, ReadSingleParameter)
#pragma alloc_text(INIT, WriteSingleParameter)
#endif

//[illegible]
//[illegible]

#pragma NDIS_INIT_FUNCTION(DriverEntry)

[illegible]
//
//[illegible]
//
[illegible]
NTSTATUS
DriverEntry(
     IN PDRIVER_OBJECT DriverObject,
     IN PUNICODE_STRING RegistryPath)
{
     NDIS_STATUS Status;
     NDIS_PROTOCOL_CHARACTERISTICS ProtocolChars;
     NDIS_MINIPORT_CHARACTERISTICS MiniportChars;
     NDIS_STRING IMName = NDIS_STRING_CONST( "IM");
     ULONG InitShutdownMask;
     PWCHAR EventLogString = IMDriverName.Buffer;
     PVOID DumpData;

#ifdef DBG
     TNSMakeBeep();
#endif
     D((0, "TNSEmul DriverEntry\n"));
     D((0, "TNSEMUL, Built %s at %s\n", __DATE__, __TIME__));


     IMDriverObject = DriverObject;


     InitializeListHead( &AdapterList );
     NdisAllocateSpinLock( &AdapterListLock );

     NdisMInitializeWrapper( &MPWrapperHandle, DriverObject, RegistryPath, NULL );

     InitShutdownMask = SHUTDOWN_TERMINATE_WRAPPER;

     Status = ConfigureDriver( RegistryPath, &ConfigData ); //[illegible]

     if ( !NT_SUCCESS( Status )) {
         D((0, "ConfigureDriver - Status: 0x%x\n", Status ));
         goto DriverEntryError;
     }


    NdisZeroMemory(&ProtocolChars, sizeof(NDIS_PROTOCOL_CHARACTERISTICS));
    ProtocolChars.Name.Length = IMName.Length;
    ProtocolChars.Name.Buffer = (PVOID)IMName.Buffer;

    ProtocolChars.MajorNdisVersion = 4;
    ProtocolChars.MinorNdisVersion = 0;

    ProtocolChars.OpenAdapterCompleteHandler = LowerMPOpenAdapterComplete;
    ProtocolChars.CloseAdapterCompleteHandler = LowerMPCloseAdapterComplete;
    ProtocolChars.SendCompleteHandler = CLSendComplete;
    ProtocolChars.TransferDataCompleteHandler = CLTransferDataComplete;
    ProtocolChars.ResetCompleteHandler = CLResetComplete;
    ProtocolChars.RequestCompleteHandler = CLRequestComplete;
    ProtocolChars.ReceiveHandler = CLReceiveIndication;
    ProtocolChars.ReceiveCompleteHandler = CLReceiveComplete;
    ProtocolChars.StatusHandler = CLStatusIndication;
    ProtocolChars.StatusCompleteHandler = CLStatusIndicationComplete;
    ~~//ProtocolChars.ReceivePacketHandler = CLReceivePacket;~~
    ProtocolChars.ReceivePacketHandler = NULL;
    ProtocolChars.BindAdapterHandler = BindToLowerMP;
    ProtocolChars.UnbindAdapterHandler = UnbindFromLowerMP;
    ProtocolChars.UnloadHandler = CLUnloadProtocol;

    NdisRegisterProtocol(&Status,
        &ClientProtocolHandle,
        &ProtocolChars,
        sizeof(NDIS_PROTOCOL_CHARACTERISTICS) + ProtocolChars.Name.Length);

    if ( !NT_SUCCESS( Status )) {
        D((0, "DoProtocolInit: couldn't register client handlers %08X\n", Status ));
    }


    if ( !NT_SUCCESS( Status )) {

        D((0, "DoProtocolInit Failed! Status: 0x%x\n", Status));

        DumpData = &Status;
        NdisWriteErrorLogEntry(IMDriverObject,
            EVENT_TRANSPORT_REGISTER_FAILED,
            TNS_ERROR_PROTOCOL_INIT,
            1,
            &EventLogString,
            sizeof( Status ),
            DumpData);

        goto DriverEntryError;
    }

    InitShutdownMask |= SHUTDOWN_DEREGISTER_PROTOCOL;

    NdisZeroMemory(&MiniportChars, sizeof(NDIS_MINIPORT_CHARACTERISTICS));
    MiniportChars.MajorNdisVersion = 4;
    MiniportChars.MinorNdisVersion = 0;

    MiniportChars.Reserved = 0;
    MiniportChars.HaltHandler = MPHalt;
    MiniportChars.InitializeHandler = MPInitialize;
    MiniportChars.QueryInformationHandler = MPQueryInformation;
    MiniportChars.ResetHandler = MPReset;
    MiniportChars.SetInformationHandler = MPSetInformation;
    MiniportChars.TransferDataHandler = MPTransferData;

    MiniportChars.ReconfigureHandler = NULL;
    MiniportChars.DisableInterruptHandler = NULL;
    MiniportChars.EnableInterruptHandler = NULL;
    MiniportChars.HandleInterruptHandler = NULL;
    MiniportChars.ISRHandler = NULL;
    MiniportChars.CheckForHangHandler = NULL;


    MiniportChars.ReturnPacketHandler = MPReturnPacket;
    MiniportChars.SendPacketsHandler = MPSendPackets;
    MiniportChars.AllocateCompleteHandler = NULL;
    MiniportChars.SendHandler = NULL;

    Status = NdisIMRegisterLayeredMiniport(MPWrapperHandle,
        &MiniportChars,
        sizeof(MiniportChars),
        &LMDriverHandle);
    if ( !NT_SUCCESS( Status )) {

        D((0, "DoMiniportInit Failed! Status: 0x%x\n", Status));

        DumpData = &Status;
        NdisWriteErrorLogEntry(IMDriverObject,
            (ULONG)TNS_EVENT_MINIPORT_REGISTER_FAILED,
            0,
            1,
            &EventLogString,
            sizeof( Status ),
            DumpData);

        goto DriverEntryError;
    }

    Status = WDMInitialize( DriverObject, &InitShutdownMask );

    if ( !NT_SUCCESS( Status )) {

        D((0, "WDMInitialize Failed! Status: 0x%x\n", Status));

        goto DriverEntryError;
    }


    return (STATUS_SUCCESS);


DriverEntryError:
    //
    if ( InitShutdownMask & SHUTDOWN_DEREGISTER_PROTOCOL ) {
        if ( ClientProtocolHandle ) {
            NdisDeregisterProtocol( &Status, ClientProtocolHandle );
            if ( Status == NDIS_STATUS_PENDING ) {
                D((0, "Client DeregProto failed - %08X\n", Status));
            }
        }
    }

    if ( InitShutdownMask & SHUTDOWN_TERMINATE_WRAPPER ) {
        NdisTerminateWrapper( MPWrapperHandle, NULL );
    }

    WDMCleanup( InitShutdownMask );

    NdisFreeSpinLock( &AdapterListLock );
    NdisFreeSpinLock( &PSAListLock );

    return (STATUS_UNSUCCESSFUL);

} // DriverEntry

VOID
CLResetComplete(
    IN  NDIS_HANDLE ProtocolBindingContext,
    IN  NDIS_STATUS Status)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
    D((0, "(%08X) CLResetComplete: Status = %08x\n", pAdapter, Status));
}

VOID
CLStatusIndication(
    IN  NDIS_HANDLE ProtocolBindingContext,
    IN  NDIS_STATUS GeneralStatus,
    IN  PVOID       StatusBuffer,
    IN  UINT        StatusBufferSize)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;

    D((0, "(%08X) CLStatusIndication: Status %08X\n", pAdapter, GeneralStatus));
    //
    // Now indicate the status to the upper layer
    //
    if (pAdapter->TNSDriverInitialized) {
        NdisMIndicateStatus( pAdapter->TNSNdisHandle, GeneralStatus, StatusBuffer, StatusBufferSize )
    }

} // CLStatusIndication
```

```
VOID
CLStatusIndicationComplete(
    IN  NDIS_HANDLE ProtocolBindingContext)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
    D((0, "(%08X) CLStatusIndicationComplete\n", pAdapter));

    if (pAdapter->TNSDriverInitialized) {
        NdisMIndicateStatusComplete(pAdapter->TNSNdisHandle);
    }
} //CLStatusIndication



NTSTATUS
ConfigureDriver (
    IN PUNICODE_STRING RegistryPath,
    IN PCONFIG_DATA ConfigurationInfo)
{
    NDIS_HANDLE ConfigHandle;
    NDIS_STATUS Status;
    NDIS_STRING TnsBlahBlah = NDIS_STRING_CONST("BlahBlah");
    PNDIS_CONFIGURATION_PARAMETER pConfigParameter;

    NdisOpenProtocolConfiguration( &Status, &ConfigHandle, RegistryPath );


    ConfigurationInfo->PacketPoolSize = 200;

    //
    // set parameters that are not dependent upon the size selected
    //

    ConfigurationInfo->DebugLevel = 10;
    ConfigurationInfo->DebugMask = 0xffffffff;

    if ( NT_SUCCESS( Status )) {

        READ_HIDDEN_CONFIG ( PacketPoolSize, NdisParameterInteger );
        NdisCloseConfiguration( ConfigHandle );
    }

    return STATUS_SUCCESS;
} //ConfigureDriver

STATIC ULONG
ReadSingleParameter(
    IN HANDLE ConfigHandle,
    IN PWCHAR ValueName,
    IN ULONG DefaultValue,
    IN NDIS_PARAMETER_TYPE NdisParamType)
{
    UNICODE_STRING ValueKeyName;
    ULONG ReturnValue;
    NDIS_STATUS Status;
    PNDIS_CONFIGURATION_PARAMETER ConfigParam;

    MyAssert( NdisParamType == NdisParameterInteger || NdisParamType == NdisParameterHexInteger );

    NdisInitUnicodeString( &ValueKeyName, ValueName );

    NdisReadConfiguration(&Status,
        &ConfigParam,
        ConfigHandle,
        &ValueKeyName,
        NdisParamType);

    if ( NT_SUCCESS( Status )) {
        ReturnValue = ConfigParam->ParameterData.IntegerData;
    } else {
        ReturnValue = DefaultValue;
    }

    return ReturnValue;
} //ReadSingleParameter

VOID
BindToLowerMP(
    OUT PNDIS_STATUS        Status,
    IN  NDIS_HANDLE         BindContext,
    IN  PNDIS_STRING        MPDeviceName,
    IN  PVOID               SystemSpecific1,
    IN  PVOID               SystemSpecific2)
{
    PADAPTER pAdapter;
    int i;
    NDIS_STATUS OpenAdapterStatus;
    NDIS_STATUS OpenErrorStatus;
    NDIS_STATUS LocalStatus;
    NDIS_MEDIUM MediumArray[] = {
        NdisMediumFddi,
        NdisMedium802_5,
        NdisMedium802_3,
        NdisMediumWan };

    UINT MediumArraySize = sizeof( MediumArray ) / sizeof( NDIS_MEDIUM );
    UINT MediaIndex;
    ULONG AdapterStructSize;
    ULONG NdisPacketTypes;
    int j;

    D((0, "BindToLowerMP: %s\n", MPDeviceName->Buffer ));

    //
    // allocate enough space for the structure and two unicode buffers to hold
    // the "IM" and underlying MP device names. We add 3 extra Unicode char to
    // to the IM device name to hold the "IM" addition to the MP name, that is be appended later
    // on and another unicode char to separate the two strings for reading.
    // The IM adapter will have the form "\Device\IM_IEEPRO3" for example, if it is sitting on top of
    // IEEPRO3
    //

    AdapterStructSize = sizeof( ADAPTER ) + MPDeviceName->Length // space for the miniport device nam
        + MPDeviceName->Length +                                 // space for the IM virtual adapter n
me: IM<MPDeviceName>)
        4 * sizeof( UNICODE_NULL );

    *Status = NdisAllocateMemory(&pAdapter, AdapterStructSize, 0, HighAddress);

    if ( pAdapter == NULL ) {
        PWCHAR StringData[2];

        StringData[0] = IMDriverName.Buffer;
        StringData[1] = L"Adapter";
        NdisWriteErrorLogEntry(IMDriverObject,
            (ULONG)EVENT_TRANSPORT_RESOURCE_POOL,
            0,
            2,
            &StringData,
            0,
            NULL);

        *Status = NDIS_STATUS_RESOURCES;
        return;
    }

    NdisZeroMemory(pAdapter, AdapterStructSize);

    GetProcessorSpeed(pAdapter);

    //
    // Get the machine name
    //
    {
        HANDLE ParamHandle;
        UNICODE_STRING KeyNameU;
        HANDLE ConfigHandle;
        ULONG Disposition;
        OBJECT_ATTRIBUTES TmpObjectAttributes;
        char nameBuf[256];
        STRING ntNameString;
        PKEY_VALUE_FULL_INFORMATION pKeyInfo;
        unsigned char keyBuffer[128];
        ULONG ResultLength;
        unsigned short *pwString;
        UNICODE_STRING ValueNameU;
        NTSTATUS Status;

        (VOID)sprintf(nameBuf, "\\Registry\\Machine\\System\\CurrentControlSet\\Control\\ComputerName
tiveComputerName");
        RtlInitString(&ntNameString, nameBuf);

        Status = RtlAnsiStringToUnicodeString(
            &KeyNameU,
                &ntNameString,
                TRUE);

        if (Status == STATUS_SUCCESS) {

            (VOID)sprintf(nameBuf, "ComputerName");
            RtlInitString(&ntNameString, nameBuf);

            Status = RtlAnsiStringToUnicodeString(
                &ValueNameU,
                &ntNameString,
                TRUE);

            InitializeObjectAttributes(
                &TmpObjectAttributes,
                &KeyNameU,
                OBJ_CASE_INSENSITIVE,
                NULL,
                NULL);

            Status = ZwCreateKey(
                &ConfigHandle,
                KEY_READ,
                &TmpObjectAttributes,
                0,
                NULL,
                0,
                &Disposition);

            Status = ZwQueryValueKey(
                ConfigHandle,
                &ValueNameU,
                KeyValueFullInformation,
                &keyBuffer,
                sizeof(keyBuffer),
                &ResultLength);

            if (Status == STATUS_SUCCESS) {
                int i;
                pKeyInfo = (PKEY_VALUE_FULL_INFORMATION) keyBuffer;

                //D((0, "pKeyInfo => %x\n", pKeyInfo));
                //D((0, "pKeyInfo->Type => %d\n", pKeyInfo->Type));
                //D((0, "pKeyInfo->NameLength => %d\n", pKeyInfo->NameLength));

                pwString = (unsigned short *)pKeyInfo;
                //D((0, "pwString => %x\n", pwString));
                pwString = (unsigned short *)((ULONG)pwString + pKeyInfo->DataOffset);
                //D((0, "pwString => %ws\n", pwString));

                i=0;
                while (*pwString && (i<MAX_COMPUTER_NAME_SIZE) ) {
                    //D((0, "*pwString => %c\n", (unsigned char) *pwString));
                    pAdapter->LocalComputerName[i++] = (unsigned char) *pwString;
                    pwString++;
                }

                D((0, "Machine Name => %s\n", pAdapter->LocalComputerName));
            }
        }
        //
        // free the unicode strings we had to allocate
        //
        RtlFreeUnicodeString(&KeyNameU);
        RtlFreeUnicodeString(&ValueNameU);
    }

    //
    // set the SMN ethernet address to broadcast for now. It will get
    // updated during the discovery/init sequence.
    //
    for (i=0; i<HARDWARE_ADDRESS_LENGTH; i++) {
        pAdapter->SMNMacAddress[i] = 0xff;
    }

    //
    // initialize the SMN Client Node tracking structure
    //
    for (i=0; i<MAX_TEAM_NODES; i++) {
        for (j=0; j<HARDWARE_ADDRESS_LENGTH; j++) {
            pAdapter->TeamNodeTable[i].TNMacAddress[j] = 0x00;
        }
        pAdapter->TeamNodeTable[i].TNNodeID = 0xffffffff;
    }

    //
    // Set adapter struct size so we know what size to use
    // later when we free it
    //
    pAdapter->AdapterStructSize = AdapterStructSize;

    //
    // Init the structures we need to manage Client and Server
    // request queues
    //

    InitializeListHead(&pAdapter->ClientWorkerListEntry);
    InitializeListHead(&pAdapter->ServerWorkerListEntry);
    InitializeListHead(&pAdapter->WorkerListEntryPool);

    KeInitializeSemaphore(&pAdapter->ClientWorkerRequestSemaphore,
        0,
        MAXLONG);
    KeInitializeSemaphore(&pAdapter->ClientWorkerResponseSemaphore,
        0,
        MAXLONG);
    KeInitializeSemaphore(&pAdapter->ServerWorkerRequestSemaphore,
        0,
        MAXLONG);
    KeInitializeSpinLock(&pAdapter->ClientWorkerListSpinLock);
    KeInitializeSpinLock(&pAdapter->ServerWorkerListSpinLock);
    KeInitializeSpinLock(&pAdapter->ListEntryPoolLock);

    KeInitializeSpinLock(&pAdapter->MyStatsLock);

    pAdapter->ListEntryItems = 50;

    for (i=0; i<(int)pAdapter->ListEntryItems; i++) {
        PREQUEST_DATA pRqstData;

        pRqstData = (PREQUEST_DATA) ExAllocatePool(NonPagedPool, sizeof(REQUEST_DATA) );

        if (pRqstData != NULL) {
            ExInterlockedInsertTailList(&pAdapter->WorkerListEntryPool,
                &pRqstData->Linkage,
                &pAdapter->ListEntryPoolLock);

        } else {
            D((0, "Cannot allocate worker queue pool\n"));
            _asm int 3
        }
    }


    //
    // make the device name point to the buffer allocated at the back of the
    // structure. The MPDevice name looks like \Device\EL165 This is the lower adapter that we want
    // to bind to. We want to construct an intermediate device that looks like \Device\IM_EL165 and [illegible]
[illegible] adapter
    // [illegible]
    //

    pAdapter->TNSDeviceName.MaximumLength = MPDeviceName->MaximumLength + 3 * sizeof( UNICODE_NULL );
[illegible]
    pAdapter->TNSDeviceName.Length = pAdapter->TNSDeviceName.MaximumLength;
    pAdapter->TNSDeviceName.Buffer = (PWSTR)( pAdapter + 1 );

    pAdapter->MPDeviceName.MaximumLength = MPDeviceName->Length;
    pAdapter->MPDeviceName.Length = pAdapter->MPDeviceName.MaximumLength;
    pAdapter->MPDeviceName.Buffer = (PWSTR)((PCHAR)pAdapter->TNSDeviceName.Buffer +
        pAdapter->TNSDeviceName.MaximumLength +
        sizeof( UNICODE_NULL ));

    // [illegible] \Device\IM_ portion
    //
    RtlCopyMemory(pAdapter->TNSDeviceName.Buffer, L"\\Device\\IM_", sizeof(L"\\Device\\IM_"));

    // [illegible]
    // [illegible]

    RtlCopyMemory(&(pAdapter->TNSDeviceName.Buffer[sizeof("\\Device\\IM")]),
        &(MPDeviceName->Buffer[sizeof("\\Device")]),
        MPDeviceName->Length - sizeof(L"\\Device"));



    //LocalStatus = GetAdapterRegistryData( (PNDIS_STRING)SystemSpecific1, Adapter );

    if (0) {
        D((0, "(%08X) BindToLowerMP: Couldn't get registry data %08X (%s)\n",
            pAdapter, LocalStatus, MPDeviceName->Buffer ));

        *Status = NDIS_STATUS_FAILURE;
        NdisFreeMemory(pAdapter, (sizeof(ADAPTER)+MPDeviceName->Length+MPDeviceName->Length+4*sizeof(UNICODE_NULL)) , 0);
        return;
    }

    //
    // Init the event now since we use it in the completion handler
    //
    // remember our binding context so we can complete BindAdapter later on
    //
    NdisInitializeEvent(&pAdapter->BlockingEvent);
    //NdisInitializeEvent(&Adapter->CLReceiveIndicationPacketCallbackEvent);
    pAdapter->BindContext = BindContext;


    // Open the adapter below us.
    NdisOpenAdapter(&OpenAdapterStatus,
        &OpenErrorStatus,
        &(pAdapter->LowerMPHandle),
        &MediaIndex,
        MediumArray,
        MediumArraySize,
        ClientProtocolHandle,
        pAdapter,
        MPDeviceName,
        0,
        NULL);


    if ( OpenAdapterStatus == NDIS_STATUS_PENDING ) {
        NdisWaitEvent( &pAdapter->BlockingEvent, 0 );
        NdisResetEvent( &pAdapter->BlockingEvent );
    } else {
        pAdapter->FinalStatus = OpenAdapterStatus;
    }

    if ( NT_SUCCESS( pAdapter->FinalStatus )) {

        pAdapter->MediaType = MediumArray[ MediaIndex ];

        if (pAdapter->MediaType = NdisMediumWan)
            pAdapter->MediaType = NdisMedium802_3;
    }
    ProcessLowerMPOpenAdapter( pAdapter, pAdapter->FinalStatus );
    pAdapter->TNSClientNodeID = 0xffffffff;

    if (TNSSharedMemoryNodeEmulation == FALSE) {
        if (PsCreateSystemThread(
            &pAdapter->ClientWorkerThreadHandle,
            (ACCESS_MASK) 0,
            (POBJECT_ATTRIBUTES) NULL,
            (HANDLE) NULL,
            (PCLIENT_ID) NULL,
            TNSClientWorkerThread,
            (PVOID) pAdapter) != STATUS_SUCCESS) {

            D((0, "Could not create client thread\n"));
            _asm int 3
        }
    } else {
        if (PsCreateSystemThread(
            &pAdapter->ServerWorkerThreadHandle,
            (ACCESS_MASK) 0,
            (POBJECT_ATTRIBUTES) NULL,
            (HANDLE) NULL,
            (PCLIENT_ID) NULL,
            TNSServerWorkerThread,
            (PVOID) pAdapter) != STATUS_SUCCESS) {

            D((0, "Could not Server worker thread\n"));
            _asm int 3
        }
    }
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c**                                                  **Pag 1 of 2**

```

    *Status = pAdapter->FinalStatus;

} //BindToLowerMP

STATIC NDIS_STATUS
GetAdapterRegistryData(
    PNDIS_STRING IMParamsKey,
    PADAPTER pAdapter)
{
    NDIS_STATUS Status;
    NDIS_HANDLE ConfigHandle;
    NDIS_STRING IMInstanceNumberKey = NDIS_STRING_CONST( "InstanceNumber" );
    PNDIS_CONFIGURATION_PARAMETER ConfigParam;

    NdisOpenProtocolConfiguration( &Status, &ConfigHandle, IMParamsKey );

    if ( !NT_SUCCESS( Status )) {
        D((0, "(%08X) GetAdapterRegistryData: can't open key %s (%08X)\n", pAdapter, IMParamsKey->Buffer,
 Status ));
        BreakPoint();
        return Status;
    }

    //
    // get the IM device instance number and build the device instance string
    //
    NdisReadConfiguration(&Status,
        &ConfigParam,
        ConfigHandle,
        &IMInstanceNumberKey,
        NdisParameterInteger);


    if ( !NT_SUCCESS( Status )) {
        D((0, "(%08X) GetAdapterRegistryData: Missing InstanceNumber key\n", pAdapter));

        Status = NDIS_STATUS_FAILURE;
        goto CloseConfig;
    }

    pAdapter->DevInstance = (USHORT)ConfigParam->ParameterData.IntegerData;

    NdisMoveMemory(pAdapter->TNSDeviceName.Buffer, IMMPName.Buffer, IMMPName.Length);

    pAdapter->TNSDeviceName.Buffer[ IMMPName.Length / sizeof( WCHAR ) ] = L'0' + pAdapter->DevInstance;


CloseConfig:
    NdisCloseConfiguration( ConfigHandle );

    return Status;

} //GetAdapterRegistryData

STATIC VOID
ProcessLowerMPOpenAdapter(
    IN  PADAPTER pAdapter,
    IN  NDIS_STATUS Status)
{
    NTSTATUS EventStatus;
    NDIS_HARDWARE_STATUS HWStatus;
    NDIS_MEDIA_STATE MediaState = 0xFFFFFFFF;
    NDIS_STRING IMDevName;
    ULONG MacOptions;
    ULONG ErrorLogData[2];
    PWCHAR StringData[2];
    PVOID DumpData;

    D((0, "(%08X) ProcessLowerMPOpenAdapter\n", pAdapter));
    //
    // [illegible] clean up and get out
    //

    if ( !NT_SUCCESS( Status )) {
        D((0, "(%08X) ProcessLowerMPOpenAdapter: binding failed %08X\n", pAdapter, Status));
        if ( Status == NDIS_STATUS_ADAPTER_NOT_FOUND ) {
            EventStatus = EVENT_TRANSPORT_ADAPTER_NOT_FOUND;
        } else {
            EventStatus = EVENT_TRANSPORT_BINDING_FAILED;
        }

```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c Page 11 of 2

```
        StringData[0] = pAdapter->TNSDeviceName.Buffer;
        StringData[1] = pAdapter->MPDeviceName.Buffer;
        DumpData = &Status;

        NdisWriteErrorLogEntry(IMDriverObject,
            EventStatus,
            0,
            2,
            &StringData,
            sizeof( Status ),
            DumpData);

        NdisFreeMemory(pAdapter, pAdapter->AdapterStructSize, 0);
        return;
    }

    D((0, "(%08X) =1 Adapter\n", pAdapter ));
    InitializeListHead( &pAdapter->ClientList );
    pAdapter->ShutdownMask = 0;


    NdisInterlockedInsertTailList(&AdapterList, &pAdapter->Linkage, &AdapterListLock);

    Status = MakeLocalNdisRequest(pAdapter,
        OID_GEN_HARDWARE_STATUS,
        &HWStatus,
        sizeof(HWStatus));

    if ( Status == NDIS_STATUS_INVALID_OID || HWStatus == NdisHardwareStatusReady ) {

        Status = MakeLocalNdisRequest(pAdapter,
            OID_GEN_MEDIA_CONNECT_STATUS,
            &MediaState,
            sizeof( MediaState ));

        if ( Status == NDIS_STATUS_INVALID_OID || MediaState == NdisMediaStateConnected ) {

            Status = MakeLocalNdisRequest(pAdapter,
                OID_GEN_LINK_SPEED,
                &pAdapter->LinkSpeed,
                sizeof( pAdapter->LinkSpeed ));

            if ( !NT_SUCCESS( Status )) {

                D((0, "(%08X) ProcessLowerMPOpenAdapter: Can't get link speed - Status %08X\n", pAdapter,
 Status));

                ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
                ErrorLogData[ 1 ] = OID_GEN_LINK_SPEED;

                NdisWriteErrorLogEntry(pAdapter->LowerMPHandle,
                    NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER,
                    2,
                    ErrorLogData);

                return;
            }

        } else {

            D((0, "(%08X) ProcessLowerMPOpenAdapter: Media not connected\n", pAdapter ));
        }
    } else {

        D((0, "(%08X) ProcessLowerMPOpenAdapter: HW Status not ready (%d)\n", HWStatus));
    }

    Status = MakeLocalNdisRequest(
        pAdapter,
        OID_802_3_CURRENT_ADDRESS,
        &pAdapter->LowerMPMacAddress,
        HARDWARE_ADDRESS_LENGTH);

    if ( NT_SUCCESS( Status )) {
        D((0, "ProcessLowerMPOpenAdapter: got hardware address => %02x %02x %02x %02x %02x %02x \n",
            pAdapter->LowerMPMacAddress[0],
            pAdapter->LowerMPMacAddress[1],
            pAdapter->LowerMPMacAddress[2],
            pAdapter->LowerMPMacAddress[3],
            pAdapter->LowerMPMacAddress[4],
            pAdapter->LowerMPMacAddress[5]));
    } else {
        D((0, "ProcessLowerMPOpenAdapter: can't get hardware address \n" ));
    }

    Status = MakeLocalNdisRequest(pAdapter,
        OID_GEN_MAC_OPTIONS,
        &MacOptions,
        sizeof(MacOptions));

    if ( NT_SUCCESS( Status )) {
        pAdapter->CopyLookaheadData = (BOOLEAN)(MacOptions & NDIS_MAC_OPTION_COPY_LOOKAHEAD_DATA);
    }

    Status = AllocatePacketPool(pAdapter);

    if (!NT_SUCCESS(Status)) {
        return;
    }

    Status = AllocateReceiveBufferPools(pAdapter);

    if (!NT_SUCCESS(Status)) {
        return;
    }

    NdisInitUnicodeString( &IMDevName, &pAdapter->TNSDeviceName.Buffer[8] );


    CurrentAdapter = pAdapter;

    D((0, "Calling NdisIMinitializeDeviceInstance\n"));
    Status = NdisIMInitializeDeviceInstance(LMDriverHandle, &IMDevName);

    if ( !NT_SUCCESS( Status )) {

        D((0, "(%08X) ProcessLowerMPOpenAdapter: can't init IM device %s (%08X)\n",
            pAdapter, IMDevName.Buffer, Status));

        ErrorLogData[ 0 ] = TNS_ERROR_CANT_INITIALIZE_IMSAMP_DEVICE;
        ErrorLogData[ 1 ] = Status;

        NdisWriteErrorLogEntry(pAdapter->LowerMPHandle,
            NDIS_ERROR_CODE_DRIVER_FAILURE,
            2,
            ErrorLogData);

        return;
    }

    pAdapter->ShutdownMask |= SHUTDOWN_DEINIT_DEV_INSTANCE;

    return;

} //ProcessLowerMPOpenAdapter

VOID
LowerMPOpenAdapterComplete(
    IN  PADAPTER pAdapter,
    IN  NDIS_STATUS Status,
    IN  NDIS_STATUS OpenErrorStatus)
{
    NDIS_MEDIA_STATE MediaState = 0xFFFFFFFF;

    D((0, "(%08X) LowerMPOpenAdapterComplete\n", pAdapter));

    pAdapter->FinalStatus = Status;
    NdisSetEvent( &pAdapter->BlockingEvent );

} //LowerMPOpenAdapterComplete

STATIC NDIS_STATUS
AllocatePacketPool(
    PADAPTER pAdapter)
{
    NDIS_STATUS Status;
    ULONG ProtoReservedSize;

    [illegible]
    //

    ProtoReservedSize = sizeof(TNS_PACKET_CONTEXT);

    NdisAllocatePacketPool(&Status,
        &pAdapter->PacketPoolHandle,
        ConfigData.PacketPoolSize,
        ProtoReservedSize);

    return Status;

}   // AllocatePacketPool

STATIC NDIS_STATUS
AllocateReceiveBufferPools(
    PADAPTER pAdapter)
{
    NDIS_STATUS Status;
    ULONG HeaderSize;
    ULONG FrameSize;    // doesn't include the header
    NDIS_ERROR_CODE ErrorCode;
    ULONG ErrorLogData[2];

    //
    // max amount of data w/o the MAC header
    //
    Status = MakeLocalNdisRequest(pAdapter,
        OID_GEN_MAXIMUM_FRAME_SIZE,
        &FrameSize,
        sizeof(FrameSize));

    if ( !NT_SUCCESS( Status )) {

        D((0, "(%08X) AllocateReceiveBufferPool: Can't get frame size - Status %08X\n", pAdapter, Status)
 );

        ErrorCode = NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER;
        ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
        ErrorLogData[ 1 ] = OID_GEN_MAXIMUM_FRAME_SIZE;
        goto ErrorExit;
    }

    //
    // this one includes the header
    //

    Status = MakeLocalNdisRequest(pAdapter,
        OID_GEN_MAXIMUM_TOTAL_SIZE,
        &pAdapter->TotalSize,
        sizeof(pAdapter->TotalSize));

    if ( !NT_SUCCESS( Status )) {

        D((0, "(%08X) AllocateReceiveBufferPool: Can't get total size - Status %08X\n", pAdapter, Status)
 );

        ErrorCode = NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER;
        ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
        ErrorLogData[ 1 ] = OID_GEN_MAXIMUM_TOTAL_SIZE;

        goto ErrorExit;
    }

    //
    // compute the real header size
    //
    HeaderSize = pAdapter->TotalSize - FrameSize;
    D((0, "FrameSize => %d, HeaderSize => %d, TotalSize => %d\n", FrameSize, HeaderSize, pAdapter->TotalS
 ize));

    Status = MakeLocalNdisRequest(pAdapter,
        OID_GEN_MAXIMUM_LOOKAHEAD,
        &pAdapter->LookaheadBufferSize,
        sizeof(pAdapter->LookaheadBufferSize));

    if ( !NT_SUCCESS( Status )) {

        D((0, "(%08X) AllocateReceiveBufferPool: Can't get lookahead size - Status %08X\n", pAdapter, Sta
 tus));

        ErrorCode = NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER;
        ErrorLogData[ 0 ] = TNS_ERROR_MISSING_OID;
        ErrorLogData[ 1 ] = OID_GEN_MAXIMUM_LOOKAHEAD;
        goto ErrorExit;
    }

    pAdapter->LookaheadBufferSize += HeaderSize;
```

File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c Page 14 of 2

```

    //
    // Init the LookaheadBuffer Pool
    //
    NdisAllocateBufferPool(&Status, &pAdapter->LookaheadPoolHandle, ConfigData.PacketPoolSize);

    return Status;

ErrorExit:

    NdisWriteErrorLogEntry(
        pAdapter->LowerMPHandle,
        ErrorCode,
        2,
        ErrorLogData );

    return Status;

}   // AllocateReceiveBufferPool


NDIS_STATUS
MPInitialize(
    OUT PNDIS_STATUS        OpenErrorStatus,
    OUT PUINT               SelectedMediumIndex,
    IN  PNDIS_MEDIUM        MediumArray,
    IN  UINT                MediumArraySize,
    IN  NDIS_HANDLE         MiniportAdapterHandle,
    IN  NDIS_HANDLE         WrapperConfigurationContext)
{
    NDIS_STRING LowerAdapterKey = NDIS_STRING_CONST( "LowerAdapter" );
    PADAPTER pAdapterInList;
    ULONG ErrorLogData[2];
    PNDIS_MINIPORT_BLOCK Mp = (PNDIS_MINIPORT_BLOCK)MiniportAdapterHandle;
    NDIS_STATUS Status;
    NDIS_HANDLE ConfigHandle;
    PNDIS_CONFIGURATION_PARAMETER pConfigParameter;
    NDIS_STRING TnsSmnModeString = NDIS_STRING_CONST("TNSSMNEmulationMode");


    D((0, "MPInitialize: Enter\n"));
    D((0, "MiniportInitialize Miniport->BaseName = %ws\n",Mp->MiniportName.Buffer ));

    pAdapterInList = FindAdapterByName(Mp->MiniportName.Buffer);


    NdisOpenConfiguration(
        &Status,
        &ConfigHandle,
        WrapperConfigurationContext);

    if (Status != STATUS_SUCCESS) {
        D((0, "Cannot open miniport config data\n"));
    } else {
        NdisReadConfiguration(
            &Status,
            &pConfigParameter,
            ConfigHandle,
            &TnsSmnModeString,
            NdisParameterHexInteger);

        if (Status != STATUS_SUCCESS) {
            D((0, "Can't read reg, Status => %x\n", Status));
        } else {
            D((0, "read reg, value => %x\n", pConfigParameter->ParameterData.IntegerData));
            TNSSharedMemoryNodeEmulation = pConfigParameter->ParameterData.IntegerData;
        }

    }

    // [illegible]
    //
    if ( !pAdapterInList ) {

        D((0, "Can't find adapter for MP dev # %ws\n",Mp->MiniportName.Buffer));

        ErrorLogData[ 0 ] = TNS_ERROR_BAD_REGISTRY_DATA;
        ErrorLogData[ 1 ] = TNS_ERROR_INVALID_IMSAMP_MP_INSTANCE;

        NdisWriteErrorLogEntry(MiniportAdapterHandle,
            NDIS_ERROR_CODE_MISSING_CONFIGURATION_PARAMETER,
            2,
                ErrorLogData);

            BreakPoint();
            return NDIS_STATUS_FAILURE;
        }

        //
        // lookup our media type in the supplied media array
        //
        for (--MediumArraySize ; MediumArraySize > 0; ) {
            if ( MediumArray[ MediumArraySize ] == pAdapterInList->MediaType ) {
                break;
            }
            if ( MediumArraySize == 0 ) {
                break;
            }
            --MediumArraySize;
        }

        if ( MediumArraySize == 0 && MediumArray[ 0 ] != pAdapterInList->MediaType ) {
            BreakPoint();
            return NDIS_STATUS_UNSUPPORTED_MEDIA;
        }

        *SelectedMediumIndex = MediumArraySize;

        //
        // save Ndis's handle to us
        //
        pAdapterInList->TNSNdisHandle = MiniportAdapterHandle;

        DM((DEBUG_INFO, DEBUG_MASKEN_INIT, "AdapterInList->TNSNdisHandle => %x\n", pAdapterInList->TNSNdisHandle));
        //
        // finish the initialization process by set our attributes
        //
        NdisMSetAttributesEx(MiniportAdapterHandle,
            pAdapterInList,
            0,
            NDIS_ATTRIBUTE_DESERIALIZE |
            NDIS_ATTRIBUTE_IGNORE_PACKET_TIMEOUT |
            NDIS_ATTRIBUTE_IGNORE_REQUEST_TIMEOUT |
            NDIS_ATTRIBUTE_INTERMEDIATE_DRIVER  ,
            0);

        //
        // keep track of being initialized
        //
        pAdapterInList->TNSDriverInitialized = TRUE;

        return NDIS_STATUS_SUCCESS;

    } // MPInitialize

    PADAPTER
    FindAdapterByName(
        PWCHAR AdapterName)
    {
        PLIST_ENTRY NextAdapter;
        PADAPTER pAdapterInList;
        ULONG NameLength = 0;
        PWCHAR pw = AdapterName;

        while ( *pw++ != 0  && NameLength < 64 ) {
            ++NameLength;
        }

        NameLength *= sizeof( WCHAR );

        NdisAcquireSpinLock( &AdapterListLock );

        NextAdapter = AdapterList.Flink;
        while ( NextAdapter != &AdapterList ) {

            pAdapterInList = CONTAINING_RECORD( NextAdapter, ADAPTER, Linkage );

            // look for the unicode null that our counting [illegible]

            if ( pAdapterInList->TNSDeviceName.Length == (NameLength+2) ) {
                if ( NdisEqualMemory(pAdapterInList->TNSDeviceName.Buffer, AdapterName, NameLength)) {
                    break;
                }
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\tnsemul.c** **Page 16 of 20**

```
        }

        NextAdapter = NextAdapter->Flink;
    }

    if ( NextAdapter != &AdapterList ) {
    } else {
        pAdapterInList = NULL;
    }

    NdisReleaseSpinLock( &AdapterListLock );

    return pAdapterInList;
}

VOID
UnbindFromLowerMP(
    OUT PNDIS_STATUS        Status,
    IN  NDIS_HANDLE         ProtocolBindingContext,
    IN  NDIS_HANDLE         UnbindContext)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
    NDIS_STATUS LocalStatus;

    D((0, "(%08X) UnbindFromLowerMP\n", pAdapter));

    if ( pAdapter->ShutdownMask & SHUTDOWN_DEINIT_DEV_INSTANCE ) {

        LocalStatus = NdisIMDeInitializeDeviceInstance(pAdapter->TNSNdisHandle);
        MyAssert(NT_SUCCESS (LocalStatus));

        pAdapter->ShutdownMask &= ~SHUTDOWN_DEINIT_DEV_INSTANCE;
    }

    pAdapter->BindContext = UnbindContext;

    *Status = NDIS_STATUS_PENDING;

} // UnbindAdapter

VOID
LowerMPCloseAdapterComplete(
    IN  NDIS_HANDLE ProtocolBindingContext,
    IN  NDIS_STATUS Status)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;

    D((0, "(%08X) LowerMPCloseAdapterComplete\n", pAdapter));

    MyAssert( NT_SUCCESS( Status ));

    if ( pAdapter->BindContext ) {
        NdisCompleteUnbindAdapter( pAdapter->BindContext, Status );
    }

    NdisAcquireSpinLock( &AdapterListLock );
    RemoveEntryList( &pAdapter->Linkage );
    NdisReleaseSpinLock( &AdapterListLock );

    if ( pAdapter->ShutdownMask & SHUTDOWN_DEALLOC_PACKET_POOL ) {

        NdisFreePacketPool( pAdapter->PacketPoolHandle );
    }

    //
    // Return Lookahead Buffer Pool Resources
    //

    if ( pAdapter->ShutdownMask & SHUTDOWN_DEALLOC_LOOKAHEAD_POOL ) {

        NdisFreeBufferPool( pAdapter->LookaheadPoolHandle );
    }


    NdisFreeSpinLock( &pAdapter->Lock );

    NdisFreeMemory(pAdapter, pAdapter->AdapterStructSize, 0);

} // LowerMPCloseAdapterComplete

VOID
CLUnloadProtocol(
    VOID)
{
    BreakPoint();
} // CLUnloadProtocol


VOID
MPHalt(
    IN  NDIS_HANDLE     MiniportAdapterContext)
{
    PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;

    D((0, "(%08X) MPHalt\n", pAdapter));
    pAdapter->ShutdownMask &= ~SHUTDOWN_DEINIT_DEV_INSTANCE;
    BreakPoint();
} // MPHalt

NDIS_STATUS
MPReset(
    OUT PBOOLEAN        AddressingReset,
    IN  NDIS_HANDLE     MiniportAdapterContext)
{
    PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
    D((0, "(%08X) MPReset\n", pAdapter));
    *AddressingReset = FALSE;
    return NDIS_STATUS_SUCCESS;
} // MPReset

//[illegible]
//
// Functions to handle making requests to the underlying miniport drivers,
// and for us to gather information on the underlying miniport drivers.
// This code is also in place to filter certain requests to/from the
// underlying miniports.
//
//[illegible]

NDIS_STATUS
MakeLocalNdisRequest(
    PADAPTER pAdapter,
    NDIS_OID Oid,
    PVOID Buffer,
    ULONG BufferSize)
{
    NDIS_STATUS Status;
    ULONG BytesNeeded, BytesWritten;

    pAdapter->Request.RequestType = NdisRequestQueryInformation;
    pAdapter->Request.DATA.QUERY_INFORMATION.Oid = Oid;
    pAdapter->Request.DATA.QUERY_INFORMATION.InformationBuffer = Buffer;
    pAdapter->Request.DATA.QUERY_INFORMATION.InformationBufferLength = BufferSize;
    pAdapter->BytesNeeded = &BytesNeeded;
    pAdapter->BytesReadOrWritten = &BytesWritten;
    pAdapter->LocalRequest = TRUE;

    NdisResetEvent( &pAdapter->BlockingEvent );

    NdisRequest(&Status, pAdapter->LowerMPHandle, &pAdapter->Request);

    //
    // only wait if the MP pended our request
    //
    if (Status == NDIS_STATUS_PENDING) {

        NdisWaitEvent( &pAdapter->BlockingEvent, 0 );
        NdisResetEvent( &pAdapter->BlockingEvent );
        Status = pAdapter->FinalStatus;
    }

    //
    // Some MPs return an NT specific status code - [illegible]
    //
    if ( Status == STATUS_NOT_SUPPORTED ) {
        Status = NDIS_STATUS_INVALID_OID;
    }

    return Status;
} // MakeLocalNdisRequest
```

```
NDIS_STATUS
MakeLocalNdisRequestSet(
    PADAPTER pAdapter,
    NDIS_OID Oid,
    PVOID Buffer,
    ULONG BufferSize)
{
    NDIS_STATUS Status;
    ULONG BytesNeeded, BytesWritten;

    pAdapter->Request.RequestType = NdisRequestSetInformation;
    pAdapter->Request.DATA.QUERY_INFORMATION.Oid = Oid;
    pAdapter->Request.DATA.QUERY_INFORMATION.InformationBuffer = Buffer;
    pAdapter->Request.DATA.QUERY_INFORMATION.InformationBufferLength = BufferSize;
    pAdapter->BytesNeeded = &BytesNeeded;
    pAdapter->BytesReadOrWritten = &BytesWritten;
    pAdapter->LocalRequest = TRUE;

    NdisResetEvent( &pAdapter->BlockingEvent );

    NdisRequest(&Status, pAdapter->LowerMPHandle, &pAdapter->Request);

    //
    // only wait if the MP pended our request
    //
    if (Status == NDIS_STATUS_PENDING) {

        NdisWaitEvent( &pAdapter->BlockingEvent, 0 );
        NdisResetEvent( &pAdapter->BlockingEvent );
        Status = pAdapter->FinalStatus;
    }

    if ( Status == STATUS_NOT_SUPPORTED ) {
        Status = NDIS_STATUS_INVALID_OID;
    }

    D((0, "MakeLocalNdisRequestSet Status => %x\n", Status));
    return Status;
} // MakeLocalNdisRequest


NDIS_STATUS
MPSetInformation(
    IN  NDIS_HANDLE     MiniportAdapterContext,
    IN  NDIS_OID        Oid,
    IN  PVOID           InformationBuffer,
    IN  ULONG           InformationBufferLength,
    OUT PULONG          BytesRead,
    OUT PULONG          BytesNeeded)
{
    PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
    NDIS_STATUS Status;
    ULONG FoundFlag;

    Status = NDIS_STATUS_FAILURE;

    D((0, "MPSetInformation, Context => %x, (%x) NDIS_OID => %s\n", pAdapter, Oid, GetNDISOidString(Oid,
&FoundFlag) ));

    // Pass on the Request and return the result
    //
    pAdapter->Request.RequestType = NdisRequestSetInformation;
    pAdapter->Request.DATA.SET_INFORMATION.Oid = Oid;
    pAdapter->Request.DATA.SET_INFORMATION.InformationBuffer = InformationBuffer;
    pAdapter->Request.DATA.SET_INFORMATION.InformationBufferLength = InformationBufferLength;
    pAdapter->BytesNeeded = BytesNeeded;
    pAdapter->BytesReadOrWritten = BytesRead;

    NdisRequest(&Status, pAdapter->LowerMPHandle, &pAdapter->Request);

    if (Status == NDIS_STATUS_SUCCESS) {
        *BytesRead = pAdapter->Request.DATA.SET_INFORMATION.BytesRead;
        *BytesNeeded = pAdapter->Request.DATA.SET_INFORMATION.BytesNeeded;
    }

    return (Status);
} // MPSetInformation


NDIS_STATUS
MPQueryInformation(
    IN  NDIS_HANDLE     MiniportAdapterContext,
    IN  NDIS_OID            Oid,
    IN  PVOID               InformationBuffer,
    IN  ULONG               InformationBufferLength,
    OUT PULONG              BytesWritten,
    OUT PULONG              BytesNeeded)
{
    PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
    NDIS_STATUS Status = NDIS_STATUS_FAILURE;
    ULONG FoundFlag;

    D((0, "MPQueryInformation, Context => %x, (%x) NDIS_OID -> %s\n", pAdapter, Oid, GetNDISOidString(Oid
 -2 , &FoundFlag) ));

    pAdapter->Request.RequestType = NdisRequestQueryInformation;
    pAdapter->Request.DATA.QUERY_INFORMATION.Oid = Oid;
    pAdapter->Request.DATA.QUERY_INFORMATION.InformationBuffer = InformationBuffer;
    pAdapter->Request.DATA.QUERY_INFORMATION.InformationBufferLength = InformationBufferLength;
    pAdapter->BytesNeeded = BytesNeeded;
    pAdapter->BytesReadOrWritten = BytesWritten;

    //
    // default case, most requests will be passed to the miniport below
    //
    NdisRequest(&Status, pAdapter->LowerMPHandle ,&pAdapter->Request);

    //
    // If the query was a success, pass the results back to the entity that made the request
    //
    if (Status == NDIS_STATUS_SUCCESS) {
        *BytesWritten = pAdapter->Request.DATA.QUERY_INFORMATION.BytesWritten;
        *BytesNeeded = pAdapter->Request.DATA.QUERY_INFORMATION.BytesNeeded;
    }

    return(Status);

} // MPQueryInformation

VOID
CLRequestComplete(
    IN  NDIS_HANDLE      ProtocolBindingContext,
    IN  PNDIS_REQUEST    NdisRequest,
    IN  NDIS_STATUS      Status)
{
    PADAPTER pAdapter = (PADAPTER) ProtocolBindingContext;
    NDIS_OID Oid = pAdapter->Request.DATA.SET_INFORMATION.Oid;
    ULONG FoundFlag;

    //
    // Complete the Set or Query, and fill in the buffer for OID_PNP_CAPABILITIES, may need be
    //
    if (pAdapter->LocalRequest) {
        pAdapter->LocalRequest = FALSE;
        NdisSetEvent(&pAdapter->BlockingEvent);
    } else {
        switch(NdisRequest->RequestType) {
            case NdisRequestQueryInformation:

                *pAdapter->BytesReadOrWritten = NdisRequest->DATA.QUERY_INFORMATION.BytesWritten;

                *pAdapter->BytesNeeded = NdisRequest->DATA.QUERY_INFORMATION.BytesNeeded;

                D((0, "CLRequest Complete, TNSNdisHandle => %x, Status => %x, (%x) Oid => %s\n",
                    pAdapter->TNSNdisHandle,
                    Status,
                    Oid,
                    GetNDISOidString(Oid, &FoundFlag)));

                NdisMQueryInformationComplete(pAdapter->TNSNdisHandle, Status);

                break;

            case NdisRequestSetInformation:

                *pAdapter->BytesReadOrWritten = NdisRequest->DATA.SET_INFORMATION.BytesRead;
                *pAdapter->BytesNeeded = NdisRequest->DATA.SET_INFORMATION.BytesNeeded;

                NdisMSetInformationComplete(pAdapter->TNSNdisHandle, Status);
                break;

            default:
                ASSERT(0);
                break;
        }
    }
} // CLRequestComplete



```

File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c Page 1 of 12

```
//*****************************************************************************
//
// COPYRIGHT:
//    This program is an unpublished work fully protected by the United
//    States copyright laws and is considered a trade secret belonging to
//    Times N Systems, Inc. To the extent that this work may be
//    considered "published," the following notice applies "1999, Times N
//    Systems, Inc." Any unauthorized use, reproduction, distribution,
//    display, modification, or disclosure of this program is strictly
//    prohibited.
//
//*****************************************************************************
//--
//*****************************************************************************
// Module:
//    recv.c - Times N intermediate driver to emulate high speed
//        interconnect
//
// Description:
//    Routines to handle receiving data, and parsing Times N specific
//    interconnect messages.
//
// Environment:
//    Windows NT Kernel Mode, Ndis Driver models
//
// Exports:
//    See Module functions generated by script processing.
//
// Author:
//    Vince Bridgers
//    vinceb@timesn.com
//--
//*****************************************************************************

#include "tns.h"
#include "tnsdebug.h"
#include "x86.h"

VOID
MPReturnPacket(
    IN  NDIS_HANDLE                 MiniportAdapterContext,
    IN  PNDIS_PACKET                Packet);

NDIS_STATUS
CLReceiveIndication(
    IN  NDIS_HANDLE                 ProtocolBindingContext,
    IN  NDIS_HANDLE                 MacReceiveContext,
    IN  PVOID                       HeaderBuffer,
    IN  UINT                        HeaderBufferSize,
    IN  PVOID                       LookAheadBuffer,
    IN  UINT                        LookaheadBufferSize,
    IN  UINT                        PacketSize);

VOID
CLReceiveComplete(
    IN  NDIS_HANDLE                 ProtocolBindingContext);

NDIS_STATUS
MPTransferData(
    OUT PNDIS_PACKET                Packet,
    OUT PUINT                       BytesTransferred,
    IN  NDIS_HANDLE                 MiniportAdapterContext,
    IN  NDIS_HANDLE                 MiniportReceiveContext,
    IN  UINT                        ByteOffset,
    IN  UINT                        BytesToTransfer);

VOID
CLTransferDataComplete(
    IN  NDIS_HANDLE                 ProtocolBindingContext,
    IN  PNDIS_PACKET                pNdisPacket,
    IN  NDIS_STATUS                 Status,
    IN  UINT                        BytesTransferred);


VOID
MPReturnPacket(
    IN  NDIS_HANDLE                 MiniportAdapterContext,
    IN  PNDIS_PACKET                Packet)
{
    PADAPTER pAdapter = (PADAPTER)MiniportAdapterContext;
    PTNS_PACKET_CONTEXT PktContext;
    PNDIS_PACKET MPPacket;
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c** **Page 2 of 12**

```
    PNDIS_BUFFER NdisBuffer;
    PBUFFER_CONTEXT BufContext;
    UINT Length;
    PUCHAR MediaArea;
    UINT Size;

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPReturnPackets =>\n"));

    //
    // See if the OriginalPacket field indicates that this belongs
    // to someone below us and return it now
    //

    PktContext = PACKET_CONTEXT_FROM_PACKET( Packet );

    MPPacket = PktContext->OriginalPacket;

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "(%08X) MPReturnPacket: IM Packet %08X\n", pAdapter, Packet));

    if ( MPPacket ) {

        D((0, "(%08X) MPReturnPacket: Returning MP Packet %08X\n", pAdapter, Packet));

        NdisReturnPackets( &MPPacket, 1 );

    } else {

        //
        // If a media specific area was allocated, free it now
        //
        NDIS_GET_PACKET_MEDIA_SPECIFIC_INFO( Packet, &MediaArea, &Size );

        NdisUnchainBufferAtFront( Packet, &NdisBuffer );

        MyAssert( NdisBuffer != NULL );

        NdisQueryBuffer( NdisBuffer, &BufContext, &Length );

        NdisFreeBuffer(NdisBuffer);
        NdisFreeMemory(BufContext, Length, 0);

        NdisUnchainBufferAtFront( Packet, &NdisBuffer );

        if ( NdisBuffer ) {
            BreakPoint();
        }
    }

    NdisReinitializePacket( Packet );
    NdisFreePacket(Packet);

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPReturnPackets <=\n"));
} // MPReturnPacket

unsigned char BroadcastAddress[] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};

int
TnsCheckAddressEtherType(
    PADAPTER pAdapter,
    unsigned char *pHeaderBuffer,
    ULONG   HeaderBufferSize)
{
    int bcast = FALSE;
    int ucast = FALSE;
    unsigned short *pEtherType;

    //
    // See if this packet is a broadcast
    //
    if (memcmp(pHeaderBuffer, BroadcastAddress, 6) == 0) {
        bcast = TRUE;
        //
        // If address is looped back (which is the case for a
        // broadcast), then do not process the packet.
        //
        if (memcmp(&pHeaderBuffer[6], pAdapter->LowerMPMacAddress, 6) == 0) {
            return FALSE;
        }
    }
    //
    // Get a pointer to the ethertype
    //
```

File: D:\nt4DDK\src\tlmesn\tnsdrvr\recv.c Page 3 of 12

```
    pEtherType = (unsigned short *)&pHeaderBuffer[12];

    //
    // If it's our packet, return true
    //
    if ( TNS_EMULATION_ETHERTYPE== wswap(*pEtherType) ) {
        return TRUE;
    }

    //
    // Otherwise it's not us
    //
    return FALSE;
}


NDIS_STATUS
CLReceiveIndication(
    IN  NDIS_HANDLE     ProtocolBindingContext,
    IN  NDIS_HANDLE     MacReceiveContext,
    IN  PVOID           HeaderBuffer,
    IN  UINT            HeaderBufferSize,
    IN  PVOID           LookaheadBuffer,
    IN  UINT            LookaheadBufferSize,
    IN  UINT            PacketSize)
{
    PADAPTER            pAdapter = (PADAPTER)ProtocolBindingContext;
    PSINGLE_LIST_ENTRY  ResidualEntry = NULL;
    PTNS_PACKET_CONTEXT PktContext;
    PNDIS_BUFFER        LookaheadNdisBuffer;
    PNDIS_PACKET        OurPacket;
    NDIS_STATUS         Status;
    NDIS_STATUS         OurPacketStatus=NDIS_STATUS_SUCCESS;
    PVOID               vBuffer;
    NDIS_PHYSICAL_ADDRESS HighAddress = NDIS_PHYSICAL_ADDRESS_CONST( -1, -1 );
    int i;

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveIndication =>\n"));


    if (!pAdapter->TNSDriverInitialized) {
        //
        // [illegible]
        //
        BreakPoint();
        return NDIS_STATUS_NOT_ACCEPTED;
    }


    //
    // snoop and spoof the packet
    //
    if (HeaderBufferSize >= 14) {
        if (TnsCheckAddressEtherType(pAdapter, HeaderBuffer, HeaderBufferSize)) {
            unsigned short *pEtherType;
            PVOID pTnsPacket = NULL;
            PTNSPacketHeader pTnsPacketHeader = NULL;
            unsigned short TNSCommand;
            //
            // [illegible]
            //
            //
            // [illegible]
            // [illegible]
            //
            // [illegible]

            if (HeaderBufferSize == PacketSize) {
                pTnsPacket = HeaderBuffer;
            }
            if ((pTnsPacket == NULL) & (HeaderBufferSize < PacketSize) ) {
                if (HeaderBufferSize == 14) {
                    pTnsPacket = &((unsigned char *)LookaheadBuffer)[-14];
                }
            }
            // [illegible]
[illegible]

            MyAssert(pTnsPacket != NULL);

            // [illegible]
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c** **Page 4 of 12**

```
            TNSCommand = wswap(((PTNSPacketHeader)pTnsPacket)->TNSCommandReply);

            switch (TNSCommand) {
                case TNS_HELLO_BROADCAST:
                    D((0, "TNS_HELLO_BROADCAST\n"));
                    if (TNSSharedMemoryNodeEmulation) {
                        //we are the smn emulator,
                        //build a reply
                        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvHelloBroadcasts);

                        //
                        //only process the reply if the shared memory region has
                        //been allocated
                        //
                        if ( (pAdapter->TNSSharedMemoryPtr) && (pAdapter->TNSSharedMemorySize) ) {
                            TNSBuildBroadcastReplyAndSend(pAdapter, pTnsPacket, HeaderBuffer);
                        }
                    } else {
                        //
                        //else just drop it - broadcast packets
                        //are looped back.
                    }
                    break;
                case TNS_HELLO_REPLY:
                    D((0, "TNS_HELLO_REPLY\n"));
                    if (TNSSharedMemoryNodeEmulation) {
                        //
                        //if we are emulating the smn, must eat
                        //this request
                        //
                        MyAssert(0);
                    } else {
                        PLIST_ENTRY pRequestObj;
                        PREQUEST_DATA pRqstData;
                        unsigned char *pBuffer;
                        //
                        //We need to get our node id and the
                        //smn mac address.
                        //
                        pAdapter->TNSClientNodeID = ((PTNSPacketHelloReply)pTnsPacket)->TNSClientNodeID;
                        D((0, "Server Hello reply, Client NodeID => %d\n", pAdapter->TNSClientNodeID));
                        pAdapter->TNSSharedMemorySize = dwswap(((PTNSPacketHelloReply)pTnsPacket)->TNSSharedMemorySize);

                        D((0, "TNSSharedMemorySize => %x\n", pAdapter->TNSSharedMemorySize));

                        for (i=0; i<6; i++) {
                            pAdapter->SMNMacAddress[i] = ((PTNSPacketHelloReply)pTnsPacket)->SMNServerMacAddress[i];
                        }
                        RtlCopyMemory(&pAdapter->SMNMachineName, ((PTNSPacketHelloReply)pTnsPacket)->SMNMachineName, 16);

                        //
                        //Dequeue a free element from our available object queue
                        //
                        pRequestObj = ExInterlockedRemoveHeadList(
                            &pAdapter->WorkerListEntryPool,
                            &pAdapter->ListEntryPoolLock);

                        pRqstData = CONTAINING_RECORD(pRequestObj,
                                    REQUEST_DATA,
                                    Linkage);

                        //
                        //[illegible]
                        //
                        pRqstData->pNdisPacket = NULL;
                        pRqstData->requestOpcode = TNS_HELLO_REPLY;

                        //
                        //Insert object into server thread object queue
                        //
                        ExInterlockedInsertTailList(
                            &pAdapter->ClientWorkerListEntry,
                            &pRqstData->Linkage,
                            &pAdapter->ClientWorkerListSpinLock);

                        //
                        //Now signal the server thread
                        //
                        KeReleaseSemaphore(
                            &pAdapter->ClientWorkerResponseSemaphore,
                            (KPRIORITY) 0,
                            (LONG) 1,
                            FALSE);
                        //
                        // We need to process this as complete
                        //

                    }
                    break;
                case TNS_READ_REQUEST:
                    //DbgPrint("TNS_READ_REQUEST\n");
                    if (TNSSharedMemoryNodeEmulation) {
                        PLIST_ENTRY pRequestObj;
                        PREQUEST_DATA pRqstData;
                        unsigned char *pBuffer;

                        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvReadRequests);

                        if (pAdapter->TNSMemoryType == VIRTUAL_MEMORY) {

                            //
                            // We need to service this read request
                            //

                            //
                            // Dequeue a free element from our available object queue
                            //
                            pRequestObj = ExInterlockedRemoveHeadList(
                                &pAdapter->WorkerListEntryPool,
                                &pAdapter->ListEntryPoolLock);

                            MyAssert(pRequestObj);

                            pRqstData = CONTAINING_RECORD(pRequestObj,
                                        REQUEST_DATA,
                                        Linkage);

                            MyAssert(pRqstData);

                            //
                            // [illegible] server thread what to do
                            //
                            pRqstData->pNdisPacket = NULL;
                            pRqstData->requestOpcode = TNS_READ_REQUEST;
                            pBuffer = (unsigned char *)&pRqstData->TnsPacket;
                            RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
                            RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSiz
e);

                            //
                            // Insert object onto server thread object queue
                            //
                            ExInterlockedInsertTailList(
                                &pAdapter->ServerWorkerListEntry,
                                &pRqstData->Linkage,
                                &pAdapter->ServerWorkerListSpinLock);

                            //
                            // Now signal the server thread
                            //
                            KeReleaseSemaphore(
                                &pAdapter->ServerWorkerRequestSemaphore,
                                (KPRIORITY) 0,
                                (LONG) 1,
                                FALSE);
                        }

                        if (pAdapter->TNSMemoryType == NONPAGED_MEMORY) {
                            PNDIS_PACKET MyPacket;
                            ULONG PacketLength;
                            PVOID pTnsBuffer;
                            NTSTATUS Status;
                            PUCHAR   vBuffer;

                            vBuffer = pAdapter->TNSSharedMemoryPtr;

                            PacketLength = TNS_PACKET_SIZE(TNSPacketReadReply);

                            Status = TNSInitializeClientNodeSendPacket(pAdapter,
                                &MyPacket,
                                &pTnsBuffer,
                                PacketLength);
```

File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c Page of 12

```
                                   RtlCopyMemory(pTnsBuffer, &((PTNSPacketHeader)pTnsPacket)->MACSrcAddress, 6);
                                   //
                                   // Stuff in relavent packet information here
                                   //
                                   ((PTNSPacketHeader)pTnsBuffer)->TNSCommandReply = wswap(TNS_READ_REPLY);

                                   ((PTNSPacketReadReply)pTnsBuffer)->RequestTag        = ((PTNSPacketReadRequest
 )pTnsPacket)->RequestTag;
                                   ((PTNSPacketReadReply)pTnsBuffer)->RequestStartTSC  = ((PTNSPacketReadRequest
 )pTnsPacket)->RequestStartTSC;
                                   vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap(((PTNSPacketReadRequest)pTnsP
 acket)->RequestOffset));

                                   if (dwswap(((PTNSPacketReadRequest)pTnsPacket)->RequestOffset) <= pAdapter->T
 NSSharedMemorySize ) {
                                       ((PTNSPacketReadReply)pTnsBuffer)->dwData = *((PULONG)vBuffer);
                                   } else {
                                       _asm int 3
                                   }
                                   TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
                               }

                           } else {
                               MyAssert(0);
                           }
                           break;
                       case TNS_READ_REPLY:
                           //DbgPrint("TNS_READ_REPLY\n");
                           if (TNSSharedMemoryNodeEmulation) {
                               //
                               // as the shm emulator, just drop this
                               //
                               MyAssert(0);
                           } else {
                               PLIST_ENTRY pRequestObj;
                               PREQUEST_DATA pRqstData;
                               unsigned char *pBuffer;
                               //
                               // We need to service this read request
                               //

                               //
                               // Dequeue a free element from our available object queue
                               //
                               pRequestObj = ExInterlockedRemoveHeadList(
                                   &pAdapter->WorkerListEntryPool,
                                   &pAdapter->ListEntryPoolLock);

                               pRqstData = CONTAINING_RECORD(pRequestObj,
                                           REQUEST_DATA,
                                           Linkage);

                               //
                               // Tell the server thread what to do
                               //
                               pRqstData->pNdisPacket = NULL;
                               pRqstData->requestOpcode = TNS_READ_REPLY;
                               pBuffer = (unsigned char *)&pRqstData->TnsPacket;
                               RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
                               RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSize);

                               //
                               // Insert the object onto server thread object queue
                               //
                               ExInterlockedInsertTailList(
                                   &pAdapter->ClientWorkerListEntry,
                                   &pRqstData->Linkage,
                                   &pAdapter->ClientWorkerListSpinLock);

                               //
                               // Signal the server thread
                               //
                               KeReleaseSemaphore(
                                   &pAdapter->ClientWorkerRequestSemaphore,
                                   (KPRIORITY) 0,
                                   (LONG) 1,
                                   FALSE);
                               //
                               // We expect to process this as complete
                               //
                           }
```

```
                    break;
                case TNS_WRITE_REQUEST:
                    //DIG(0, "TNS WRITE REQUEST\n");

                    if (TNSSharedMemoryNodeEmulation) {

                        TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvWriteRequests);

                        if (pAdapter->TNSMemoryType == VIRTUAL_MEMORY) {
                            //
                            // Process this request on the smm
                            //
                            PLIST_ENTRY pRequestObj;
                            PREQUEST_DATA pRqstData;
                            unsigned char *pBuffer;
                            //
                            // We need to service this read request
                            //

                            //
                            // Dequeue a free element from our available object queue
                            //
                            pRequestObj = ExInterlockedRemoveHeadList(
                                &pAdapter->WorkerListEntryPool,
                                &pAdapter->ListEntryPoolLock);

                            pRqstData = CONTAINING_RECORD(pRequestObj,
                                        REQUEST_DATA,
                                        Linkage);

                            //
                            // [illegible]
                            //
                            pRqstData->pNdisPacket = NULL;
                            pRqstData->requestOpcode = TNS_WRITE_REQUEST;
                            pBuffer = (unsigned char *)&pRqstData->TnsPacket;
                            RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
                            RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSize);

                            //
                            // Insert element onto server thread object queue
                            //
                            ExInterlockedInsertTailList(
                                &pAdapter->ServerWorkerListEntry,
                                &pRqstData->Linkage,
                                &pAdapter->ServerWorkerListSpinLock);

                            //
                            // Now signal the server thread
                            //
                            KeReleaseSemaphore(
                                &pAdapter->ServerWorkerRequestSemaphore,
                                (KPRIORITY) 0,
                                (LONG) 1,
                                FALSE);
                        }

                        if (pAdapter->TNSMemoryType == NONPAGED_MEMORY) {

                            PNDIS_PACKET MyPacket;
                            ULONG PacketLength;
                            PVOID pTnsBuffer;
                            NTSTATUS Status;
                            PUCHAR   vBuffer;

                            [illegible]

                            vBuffer = pAdapter->TNSSharedMemoryPtr;

                            vBuffer = (PUCHAR)((ULONG)vBuffer+(ULONG)dwswap( ((PTNSPacketWriteRequest)pTnsPacket)->RequestOffset));

                            if (dwswap( ((PTNSPacketWriteRequest)pTnsPacket)->RequestOffset) <= pAdapter->TNSSharedMemorySize ) {

                                *((PULONG)vBuffer) = ((PTNSPacketWriteRequest)pTnsPacket)->dwData;
                            } else {
                                _asm int 3
                            }

                            //
                            // [illegible]
```

```
                                    //

                                    PacketLength = TNS_PACKET_SIZE(TNSPacketWriteReply);
                                    Status = TNSInitializeClientNodeSendPacket(pAdapter,
                                         &MyPacket,
                                         &pTnsBuffer,
                                         PacketLength);

                                    RtlCopyMemory(pTnsBuffer, &((PTNSPacketWriteRequest)pTnsPacket)->MACSrcAddress, 6);

                                    //
                                    //Fill in relevant packet information here
                                    //
                                    ((PTNSPacketWriteReply)pTnsBuffer)->TNSCommandReply = wswap(TNS_WRITE_ACK);
                                    ((PTNSPacketWriteReply)pTnsBuffer)->RequestTag      = ((PTNSPacketWriteRequest)pTnsPacket)->RequestTag;
                                    ((PTNSPacketWriteReply)pTnsBuffer)->RequestStartTSC = ((PTNSPacketWriteRequest)pTnsPacket)->RequestStartTSC;

                                    TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);
                               }

                         } else {
                               //
                               //drop this - we are not the [illegible]
                               //
                               MyAssert(0);
                         }
                         break;

                    case TNS_WRITE_ACK:
                         //[illegible] TNS_WRITE_ACK [illegible]
                         if (TNSSharedMemoryNodeEmulation) {
                               //
                               //are the smd emulator, just drop this
                               //
                               MyAssert(0);
                         } else {
                               PLIST_ENTRY pRequestObj;
                               PREQUEST_DATA pRqstData;
                               unsigned char *pBuffer;
                               //
                               //We must service this read request
                               //


                               //
                               //Dequeue a free element from our available object queue
                               //
                               pRequestObj = ExInterlockedRemoveHeadList(
                                    &pAdapter->WorkerListEntryPool,
                                    &pAdapter->ListEntryPoolLock);

                               pRqstData = CONTAINING_RECORD(pRequestObj,
                                           REQUEST_DATA,
                                           Linkage);

                               //
                               //[illegible] do
                               //
                               pRqstData->pNdisPacket = NULL;
                               pRqstData->requestOpcode = TNS_WRITE_ACK;
                               pBuffer = (unsigned char *)&pRqstData->TnsPacket;
                               RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
                               RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSize);

                               //
                               //Insert object onto the worker thread object queue
                               //
                               ExInterlockedInsertTailList(
                                    &pAdapter->ClientWorkerListEntry,
                                    &pRqstData->Linkage,
                                    &pAdapter->ClientWorkerListSpinLock);

                               //
                               //[illegible]
                               //
                               KeReleaseSemaphore(
                                    &pAdapter->ClientWorkerRequestSemaphore,
                                    (KPRIORITY) 0,
                                    (LONG) 1,
                                    FALSE);
                               //
                    // We need to process this as complete
                    //
                }
                break;
            case TNS_QUERY_STATS: {
                [illegible]
                    PLIST_ENTRY pRequestObj;
                    PREQUEST_DATA pRqstData;
                    unsigned char *pBuffer;

                    PNDIS_PACKET MyPacket;
                    ULONG PacketLength;
                    PTNSPacketQueryStatsReply pTnsBuffer;
                    NTSTATUS Status;
                    NDIS_STATUS NdisStatus;
                    PUCHAR   vBuffer;

                    TnsIncrementStat(pAdapter, &pAdapter->MyStats.numSrvQueryStats);

                    vBuffer = pAdapter->TNSSharedMemoryPtr;

                    PacketLength = TNS_PACKET_SIZE(TNSPacketQueryStatsReply);

                    Status = TNSInitializeClientNodeSendPacket(pAdapter,
                         &MyPacket,
                         &pTnsBuffer,
                         PacketLength);

                    RtlCopyMemory(pTnsBuffer, &((PTNSPacketHeader)pTnsPacket)->MACSrcAddress, 6);
                    //
                    // [illegible] packet information here
                    //
                    pTnsBuffer->TNSCommandReply = wswap(TNS_QUERY_STATS_REPLY);

                    pTnsBuffer->RequestTag = ((PTNSPacketQueryStats)pTnsPacket)->RequestTag;
                    pTnsBuffer->RequestStartTSC = ((PTNSPacketQueryStats)pTnsPacket)->RequestStartTSC;


                    RtlCopyMemory(&pTnsBuffer->TnsNodeStatistics, &pAdapter->MyStats, sizeof(STATISTICS) );

                    RtlCopyMemory(&pTnsBuffer->MpStats, &pAdapter->mpStats, sizeof(MPSTATS) );

                    pTnsBuffer->NdisStatus = STATUS_SUCCESS;

                    TNSSendPackets(pAdapter->LowerMPHandle, &MyPacket, 1);

                }
                break;

            case TNS_CLEAR_STATS:
                [illegible]

                RtlZeroMemory(&pAdapter->MyStats, sizeof(STATISTICS) );
                RtlZeroMemory(&pAdapter->mpStats, sizeof(MPSTATS) );

                break;

            case TNS_QUERY_STATS_REPLY: {
                [illegible]
                    PLIST_ENTRY pRequestObj;
                    PREQUEST_DATA pRqstData;
                    unsigned char *pBuffer;
                    //
                    // [illegible]
                    //


                    //
                    // [illegible]
                    //
                    pRequestObj = ExInterlockedRemoveHeadList(
                         &pAdapter->WorkerListEntryPool,
                         &pAdapter->ListEntryPoolLock);

                    pRqstData = CONTAINING_RECORD(pRequestObj,
                                 REQUEST_DATA,
                                 Linkage);


                    //
                    // [illegible]
                    //
                    pRqstData->pNdisPacket = NULL;
                    pRqstData->requestOpcode = TNS_QUERY_STATS_REPLY;
```

```
                        pBuffer = (unsigned char *)&pRqstData->TnsPacket;
                        RtlCopyMemory(pBuffer, HeaderBuffer, HeaderBufferSize);
                        RtlCopyMemory(&pBuffer[HeaderBufferSize], LookaheadBuffer, LookaheadBufferSize);

                        //
                        // Insert object onto server thread object queue
                        //
                        ExInterlockedInsertTailList(
                            &pAdapter->ClientWorkerListEntry,
                            &pRqstData->Linkage,
                            &pAdapter->ClientWorkerListSpinLock);

                        //
                        // Now signal the Server thread
                        //
                        KeReleaseSemaphore(
                            &pAdapter->ClientWorkerRequestSemaphore,
                            (KPRIORITY) 0,
                            (LONG) 1,
                            FALSE);
                        //
                        // We need to process this as complete
                        //
                    }
                    break;

                case TNS_STRING_WRITE_REQUEST:
                    D((0, "TNS_STRING_WRITE_REQUEST\n"));
                    MyAssert(0);
                    if (TNSSharedMemoryNodeEmulation) {
                    } else {
                    }
                    break;
                case TNS_STRING_READ_REQUEST:
                    D((0, "TNS_STRING_READ_REQUEST\n"));
                    MyAssert(0);
                    if (TNSSharedMemoryNodeEmulation) {
                    } else {
                    }
                    break;
                case TNS_STRING_READ_REPLY:
                    D((0, "TNS_STRING_READ_REPLY\n"));
                    MyAssert(0);
                    if (TNSSharedMemoryNodeEmulation) {
                    } else {
                    }
                    break;
                default:
                    D((0, "Unrecognized command => %x\n", TNSCommand));
                    D((0, "HeaderBuffer   => %x, HdrBufferSize => %x\n", HeaderBuffer, HeaderBufferSize));
                    D((0, "LookahedBuffer => %x, LABufferSize  => %x\n", LookaheadBuffer, LookaheadBufferSize));
                    MyAssert(0);
                    break;
            }
            //return NDIS_STATUS_SUCCESS;
        }

    } else {
        D((0, "HeaderBufferSize not equal to or gt than 14, HeaderBufferSize => %d\n", HeaderBufferSize));
        _asm int 3
    }

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "HeaderBuffer => %x, HeaderBufferSize => %x, LookaheadBuffer => %x, LookaheadBufferSize => %x\n",
         HeaderBuffer,
         HeaderBufferSize,
         LookaheadBuffer,
         LookaheadBufferSize));

    NdisAllocatePacket(&Status, &OurPacket, pAdapter->PacketPoolHandle);

    NdisReinitializePacket(OurPacket);

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "CLReceiveIndication: OurPacket => %x\n", OurPacket));

    MyAssert(OurPacket->Private.Head == NULL);

    NDIS_SET_PACKET_STATUS(OurPacket, OurPacketStatus);

    Status = NdisAllocateMemory(&vBuffer, 2000, 0, HighAddress);

    if (Status != NDIS_STATUS_SUCCESS) {
        BreakPoint();
    }

    NdisAllocateBuffer(&Status,
        &LookaheadNdisBuffer,
        pAdapter->LookaheadPoolHandle,
        vBuffer,
        2000);

    if (Status != NDIS_STATUS_SUCCESS) {
        BreakPoint();
    }

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "CLReceiveIndication: LookaheadNdisBuffer => %x\n", LookaheadNdisBuffer));

    PktContext = PACKET_CONTEXT_FROM_PACKET(OurPacket);

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "(%08X) CLReceiveIndication: Packet %08X Packetsize %d %s\n",
       pAdapter, OurPacket, PacketSize,
       (PacketSize != LookaheadBufferSize ? "(RD)" : "")));

    PktContext->OriginalPacket = NULL;

    if (pAdapter->CopyLookaheadData) {
        NdisMoveMemory(vBuffer, HeaderBuffer, HeaderBufferSize);
        NdisMoveMemory((CHAR *)vBuffer+HeaderBufferSize, LookaheadBuffer, LookaheadBufferSize);
    } else {
        TdiCopyLookaheadData(vBuffer, HeaderBuffer, HeaderBufferSize, 0);
        TdiCopyLookaheadData((CHAR *)vBuffer+HeaderBufferSize, LookaheadBuffer, LookaheadBufferSize, 0);
    }

    NdisAdjustBufferLength(LookaheadNdisBuffer, HeaderBufferSize+LookaheadBufferSize);
    NDIS_SET_PACKET_HEADER_SIZE(OurPacket, HeaderBufferSize);
    NdisChainBufferAtFront(OurPacket, LookaheadNdisBuffer);

    DUMP_PACKET(OurPacket);

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "Adapter->TNSNdisHandle => %x, OurPacket => %x\n", pAdapter->TNSNdisHandle, OurPacket));
    NDIS_SET_PACKET_STATUS(OurPacket, NDIS_STATUS_RESOURCES);

    NdisMIndicateReceivePacket(pAdapter->TNSNdisHandle, &OurPacket, 1);

    if ( NDIS_GET_PACKET_STATUS(OurPacket) != NDIS_STATUS_PENDING) {
        MPReturnPacket( (NDIS_HANDLE)pAdapter, OurPacket );
    }

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveIndication <=\n"));
    return NDIS_STATUS_SUCCESS;

} //CLReceiveIndication


VOID
CLReceiveComplete(
    IN  NDIS_HANDLE     ProtocolBindingContext)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveComplete =>\n"));

    if (pAdapter->TNSDriverInitialized) {

        switch( pAdapter->MediaType ) {
            case NdisMedium802_3:
                DM((DEBUG_VERBOSE, DEBUG_MASKEN_RECV, "(%08X) CLReceiveComplete: 802_3\n", pAdapter));
                NdisMEthIndicateReceiveComplete( pAdapter->TNSNdisHandle );
                break;

            case NdisMedium802_5:
                D((0, "(%08X) CLReceiveComplete: 802_5\n", pAdapter));
                BreakPoint();
                NdisMTrIndicateReceiveComplete( pAdapter->TNSNdisHandle );
                break;

            case NdisMediumFddi:
                D((0, "(%08X) CLReceiveComplete: FDDI\n", pAdapter));
                BreakPoint();
```

File: D:\nt4DDK\src\timesn\tnsdrvr\recv.c Page 12 of 12

```
                NdisMFddiIndicateReceiveComplete( pAdapter->TNSNdisHandle );
                break;

            default:
                MyAssert( FALSE );
        }
    } else {
        BreakPoint();
    }

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLReceiveComplete <=\n"));
} // CLReceiveComplete

NDIS_STATUS
MPTransferData(
    OUT PNDIS_PACKET        Packet,
    OUT PUINT               BytesTransferred,
    IN  NDIS_HANDLE         MiniportAdapterContext,
    IN  NDIS_HANDLE         MiniportReceiveContext,
    IN  UINT                ByteOffset,
    IN  UINT                BytesToTransfer)
{
    PADAPTER Adapter = (PADAPTER)MiniportAdapterContext;

    D((0, "(%08X) MPTransferData:\n", Adapter));
    BreakPoint();
    return NDIS_STATUS_FAILURE;
} // MPTransferData

VOID
CLTransferDataComplete(
    IN  NDIS_HANDLE         ProtocolBindingContext,
    IN  PNDIS_PACKET        Packet,
    IN  NDIS_STATUS         Status,
    IN  UINT                BytesTransferred)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
    PTNS_PACKET_CONTEXT PktContext;

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLTransferComplete =>\n"));
    D((0, "(%08X) CLTransferDataComplete: Packet %08X Status %08X Bytes xfer'ed %d\n",
        pAdapter, Packet, Status, BytesTransferred));

    PktContext = PACKET_CONTEXT_FROM_PACKET( Packet );

    NdisChainBufferAtFront(Packet, PktContext->LookaheadBuffer);

    NdisMIndicateReceivePacket(pAdapter->TNSNdisHandle, &Packet, 1);

    if ( NDIS_GET_PACKET_STATUS(Packet) != NDIS_STATUS_PENDING) {
        MPReturnPacket((NDIS_HANDLE)pAdapter, Packet);
    }

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLTransferComplete <=\n"));
} // CLTransferDataComplete
```

File: D:\nt4DDK\src\timesn\tnsdrvr\send.c Page 1 of 3

```
//*****************************************************************************
//
// COPYRIGHT:
//      This program is an unpublished work fully protected by the United
//      States copyright laws and is considered a trade secret belonging to
//      Times N Systems, Inc.  To the extent that this work may be
//      considered "published," the following notice applies: "1999, Times N
//      Systems, Inc."  Any unauthorized use, reproduction, distribution,
//      display, modification, or disclosure of this program is strictly
//      prohibited.
//
//*****************************************************************************
//
//*****************************************************************************
// Module:
//      send.c - Module to handle sending packets.
//
// Description:
//
// Environment:
//      Windows NT Kernel mode, Ndis Driver models.
//
// Exports:
//      See Module functions generated by script processing.
//
// Author:
//      Vince Bridges
//      vinceb@timesn.com
//
//
//*****************************************************************************

#include "tns.h"
#include "tnsdebug.h"
#include "x86.h"

#define MAX_LOCAL_PACKET_ARRAY  10

VOID
MPSendPackets(
     IN  NDIS_HANDLE MiniportAdapterContext,
     IN  PPNDIS_PACKET PacketArray,
     IN  UINT NumberOfPackets
     );


VOID
CLSendComplete(
     IN  NDIS_HANDLE ProtocolBindingContext,
     IN  PNDIS_PACKET Packet,
     IN  NDIS_STATUS Status
     );

VOID
MPSendPackets(
     IN  NDIS_HANDLE      MiniportAdapterContext,
     IN  PPNDIS_PACKET    PacketArray,
     IN  UINT             NumberOfPackets)
{
     PADAPTER         pAdapter=(PADAPTER)MiniportAdapterContext;
     PNDIS_PACKET     Packet;
     PNDIS_PACKET     MyPacket;
     PNDIS_PACKET     MyPacketArray[MAX_LOCAL_PACKET_ARRAY];

     PSINGLE_LIST_ENTRY       PacketEntry = NULL;
     PTNS_PACKET_CONTEXT      PktContext;
     PNDIS_BUFFER     FirstBuffer;
     PNDIS_PACKET_OOB_DATA    MyOOBData;
     PNDIS_PACKET_OOB_DATA    OOBData;
     ULONG         PacketLength, i;
     ULONG         NumMyPackets=0;
     NDIS_STATUS         Status;

     DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPSendPackets =>\n"));
     DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "(%08X) MPSendPackets: %d XPORT packets\n", pAdapter, Numb
ackets));

     if (pAdapter) {
          if (!pAdapter->TNSDriverInitialized) {
               //
               // Need to go through all packets, marking them as failed
               //
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\send.c** **Page 2 of 3**

```
            BreakPoint();
        }
    }

    for (i=0; i<NumberOfPackets; ++i) {     //process the whole array of packets

        //pointer to the packet we are working on

        Packet = PacketArray[i];

        DUMP_PACKET(Packet);
        //
        // Initialize our context area, chain the buffers from the
        // upper layer packet and other junk like packet flags
        //
        NdisAllocatePacket(&Status, &MyPacket, pAdapter->PacketPoolHandle);

        //
        // make sure ptrs are valid
        //
        MyAssert(MyPacket->Private.Head == NULL);

        PktContext = PACKET_CONTEXT_FROM_PACKET(MyPacket);

        DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "MPSendPackets: MyPacket => %x\n", PacketEntry));


        NdisQueryPacket(Packet, NULL, NULL, &FirstBuffer, &PacketLength);

        NdisChainBufferAtFront(MyPacket, FirstBuffer);

        NdisSetPacketFlags(MyPacket, NdisGetPacketFlags(Packet));

        OOBData = NDIS_OOB_DATA_FROM_PACKET(Packet);
        MyOOBData = NDIS_OOB_DATA_FROM_PACKET(MyPacket);
        NdisMoveMemory(MyOOBData, OOBData, sizeof(NDIS_PACKET_OOB_DATA));

        //
        // [illegible]
        //
        NDIS_SET_PACKET_STATUS(Packet, NDIS_STATUS_PENDING);

        //
        // prepare our context area so we know what to do when it
        // [illegible] completed
        //
        PktContext->OriginalPacket = Packet;
        PktContext->SMNEmulationPacket = FALSE;

        //
        // add to the array of packets we are going to send out immediately
        //
        DUMP_PACKET(MyPacket);
        MyPacketArray[NumMyPackets++] = MyPacket;
    }

    if (NumMyPackets) {
        int FoundFlag;
        for (i=0; i<NumMyPackets; i++) {
            DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "MPSendPackets, Packet Status => %x, %s\n",
            NDIS_GET_PACKET_STATUS(MyPacketArray[i]),
            GetNDISStatusString(NDIS_GET_PACKET_STATUS(MyPacketArray[i]), &FoundFlag) ));
        }
        NdisSendPackets(pAdapter->LowerMPHandle, &MyPacketArray[0], NumMyPackets);
    }

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "MPSendPackets <=\n"));
} //MPSendPackets

int printbuftime = 1;

VOID
CLSendComplete(
    IN  NDIS_HANDLE      ProtocolBindingContext,
    IN  PNDIS_PACKET     Packet,
    IN  NDIS_STATUS      Status)
{
    PADAPTER pAdapter = (PADAPTER)ProtocolBindingContext;
    PTNS_PACKET_CONTEXT PktContext;
    int FoundFlag;
    int SMNEmulationPacket;
    PNDIS_BUFFER MyBuffer;
```

**File: D:\nt4DDK\src\timesn\tnsdrvr\send.c** **Page 3 of 3**

```
    PTNSPacketReadRequest BufContext;
    UINT Length;

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLSendComplete =>\n"));

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "CLSendComplete, Packet Status => %x, %s\n",
        NDIS_GET_PACKET_STATUS(Packet),
        GetNDISStatusString(NDIS_GET_PACKET_STATUS(Packet), &FoundFlag) ));

    PktContext = PACKET_CONTEXT_FROM_PACKET(Packet);
    SMNEmulationPacket = PktContext->SMNEmulationPacket;

    DUMP_PACKET(Packet);
    if (PktContext->OriginalPacket) {
        DUMP_PACKET(PktContext->OriginalPacket);
        DM((DEBUG_VERBOSE, DEBUG_MASKEN_SEND, "CLSendComplete, Packet Status => %x, %s\n",
            NDIS_GET_PACKET_STATUS(PktContext->OriginalPacket),
            GetNDISStatusString(NDIS_GET_PACKET_STATUS(PktContext->OriginalPacket), &FoundFlag) ));
    }

    if (SMNEmulationPacket) {
        NdisUnchainBufferAtFront(Packet, &MyBuffer);
        NdisQueryBuffer(MyBuffer, &BufContext, &Length);
        NdisFreeBuffer(MyBuffer);
        NdisFreeMemory(BufContext, Length, 0);
    }

    // BUGBUG: Do I have to unchain and free the buffers before freeing this?
    NdisReinitializePacket(Packet);
    NdisFreePacket(Packet);
    // and complete the orig. packet
    //

    if (SMNEmulationPacket == FALSE) {
        NdisMSendComplete(pAdapter->TNSNdisHandle, PktContext->OriginalPacket, Status);
    }

    DM((DEBUG_VERBOSE, DEBUG_MASKEN_ENTRYEXIT, "CLSendComplete <=\n"));
} // CLSendComplete
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.h**

```
//==========================================================================
//
// COPYRIGHT:
//    This program is an unpublished work fully protected by the United
//    States copyright laws and is considered a trade secret belonging to
//    Times N Systems, Inc.  To the extent that this work may be
//    considered "published," the following notice applies: "1999, Times N
//    Systems, Inc."  Any unauthorized use, reproduction, distribution,
//    display, modification, or disclosure of this program is strictly
//    prohibited.
//
//==========================================================================
//
//==========================================================================
// Module:
//
// Description:
//
// Environment:
//    Windows NT Kernel Mode only.
//
// Exports:
//    See Module functions generated by script processing.
//
// Author:
//    Vince Bridgers
//    vinceb@timesn.com
//
//==
//==========================================================================



//
// Define the various device type values.  Note that values used by Microsoft
// Corporation are in the range 0-32767, and 32768-65535 are reserved for use
// by customers.
//

#define FILE_DEVICE_TNSCLIENT  0x00008300



//
// Macro definition for defining IOCTL and FSCTL function control codes.  Note
// that function codes 0-2047 are reserved for Microsoft Corporation, and
// 2048-4095 are reserved for customers.
//

#define TNSCLIENT_IOCTL_INDEX  0x830



#define IOCTL_TNSCLIENT_HELLO           CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
                                                 TNSCLIENT_IOCTL_INDEX,  \
                                                 METHOD_BUFFERED,        \
                                                 FILE_ANY_ACCESS)

#define IOCTL_TNSCLIENT_GET_LOCAL_STATS CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
                                                 TNSCLIENT_IOCTL_INDEX+1, \
                                                 METHOD_BUFFERED,        \
                                                 FILE_ANY_ACCESS)

#define IOCTL_TNSCLIENT_GET_SMN_STATS   CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
                                                 TNSCLIENT_IOCTL_INDEX+2, \
                                                 METHOD_BUFFERED,        \
                                                 FILE_ANY_ACCESS)


#define IOCTL_TNSCLIENT_GET_SMN_INFO    CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
                                                 TNSCLIENT_IOCTL_INDEX+3, \
                                                 METHOD_BUFFERED,        \
                                                 FILE_ANY_ACCESS)

#define IOCTL_TNSCLIENT_GET_LOCAL_INFO  CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
                                                 TNSCLIENT_IOCTL_INDEX+4, \
                                                 METHOD_BUFFERED,        \
                                                 FILE_ANY_ACCESS)


#define IOCTL_TNSCLIENT_DOTEST          CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
                                                        TNSCLIENT_IOCTL_INDEX+5,  \
                                                        METHOD_BUFFERED,         \
                                                        FILE_ANY_ACCESS)

#define IOCTL_TNSCLIENT_CLEAR_STATS      CTL_CODE(FILE_DEVICE_TNSCLIENT,  \
                                                        TNSCLIENT_IOCTL_INDEX+6,  \
                                                        METHOD_BUFFERED,         \
                                                        FILE_ANY_ACCESS)


#define IOCTL_TNSCLIENT_GET_SMN_TABLE_INFO CTL_CODE(FILE_DEVICE_TNSCLIENT,   \
                                                        TNSCLIENT_IOCTL_INDEX+7,  \
                                                        METHOD_BUFFERED,         \
                                                        FILE_ANY_ACCESS)

#define IOCTL_TNSCLIENT_GET_NODE_INFO    CTL_CODE(FILE_DEVICE_TNSCLIENT, \
                                                        TNSCLIENT_IOCTL_INDEX+8, \
                                                        METHOD_BUFFERED, \
                                                        FILE_ANY_ACCESS)





#define ETHERNET_ADDRESS_LEN 6
#define MAX_COMPUTER_NAME_LEN   16
//
// Data structure for packet of data [illegible]
//
typedef struct _IODRIVER_PACKET {
    int     MaxNumWrites;
    int     MaxNumReads;
    int     MaxNumReadWrites;

    STATISTICS  Stats;
    MPSTATS     MpStats;

    unsigned char MacAddress[ETHERNET_ADDRESS_LEN];
    unsigned char ComputerName[MAX_COMPUTER_NAME_LEN];
    unsigned long TeamNodeID;
    unsigned long TNSSharedMemorySize;

    unsigned long TestStatus;

    unsigned long DebugPrintFlag;
    unsigned long DebugPrintMask;

    SMNTableInfo  SMNInfo[MAX_TEAM_NODES];

} IO_DRIVER_PACKET, *pIO_DRIVER_PACKET;
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c** **Page 1 of 9**

```
//***************************************************************************
//
// COPYRIGHT:
//     This program is an unpublished work fully protected by the United
//     States copyright laws and is considered a trade secret belonging to
//     Times N Systems, Inc. To the extent that this work may be
//     considered "published," the following notice applies "  1999, Times N
//     Systems, Inc." Any unauthorized use, reproduction, distribution,
//     display, modification, or disclosure of this program is strictly
//     prohibited.
//
//****************************************************************************
//--
//***************************************************************************
// Module:
//
// Description:
//
// Environment:
//     Windows NT Kernel Mode only.
//
// Exports:
//
// Author:
//     Vince Bridgers
//     vinceb@timesn.com
//
//==
//****************************************************************************

#include <ntddk.h>
#include <stdarg.h>
#include <stdio.h>
#include "tnsstats.h"
#include "tnsclien.h"
#include "x86.h"

//
// A structure representing the instance information associated with
// a particular device
//

typedef struct _DEVICE_EXTENSION {
    ULONG  StateVariable;
} DEVICE_EXTENSION, *PDEVICE_EXTENSION;


VOID GetSidt(PVOID);


ULONG GTestFlag=10;
ULONG _gPrintStats = 0;


extern unsigned char *MyTrap0E;


NTSTATUS
TNSClientDrvDispatch(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP Irp
    );

VOID
TNSClientDrvUnload(
    IN PDRIVER_OBJECT DriverObject
    );

ULONG PFPrintFlag = FALSE;

#define TESTTIMES   1000

//---------------------------------------------------------------------------
//
// Linear congruent pseudorandom number generator for initialization
//
//---------------------------------------------------------------------------
unsigned long seed=1;

//---------------------------------------------------------------------------
//
// Return a pseudorandom number in the interval 0 <= n < 32768
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c** **Page 2 of 9**

```
// This produces the following sequence of pseudorandom numbers:
// 346, 130, 10982, 1090, ... (9996 numbers skipped) ..., 23369,
// 2020, 5703, 12762, 10828, 16252, 28648, 27041, 23444, 6604, ...
//
//--------------------------------------------------------------------------

//**************************************************************************
//--
unsigned
myrand()
//
// Description:
//     Return a 16 bit random number from a linear congruent pseudorandom
//     number generator in the range 0 <= n < 32768.
//--
//**************************************************************************
{
    seed = seed*0x015a4e35L + 1;
    return (seed>>16)&0x7fff;
}

//**************************************************************************
//--
unsigned long
myrand32()
//
// Description:
//     Return a 32-bit random number from a linear congruent pseudorandom
//     number generator in the range 0 <= n < 2^32.
//--
//**************************************************************************
{
    unsigned long n;
    n = myrand();
    n = n << 16;
    n |= myrand();
    return n;
}

//**************************************************************************
//--
unsigned long
myrand32n(unsigned long clipvalue)
//
// Description:
//     Return a 32-bit random number from a linear congruent pseudorandom
//     number generator in the range 0 <= n < 2^32.
//--
//**************************************************************************
{
    unsigned long n;
    n = myrand();
    n = n << 16;
    n |= myrand();

    if (clipvalue == 0)
        return 1;

    return (n % clipvalue);
}


//**************************************************************************
//--
unsigned
myrandn(
    unsigned n)     // clip number
//
// Description:
//     Return a clipped random number. This function returns a random
//     number in the range 0 <= randn < n.
//--
//**************************************************************************
{
    if (n == 0)
        return 1;

    return (myrand() % n);
}

//--------------------------------------------------------------------------
//
```

```
// Initialize above linear congruent pseudorandom number generator.
//
//[illegible]


//[illegible]
//--
void
mysrand(
    unsigned newseed)
//
// Description:
//      Sets the random number generator seed to a new value.
//--
//[illegible]
{
    seed = newseed;
}

//[illegible]
//--
unsigned
getseed(void)
//
// Description:
//      Gets the current random number generator seed.
//--
//[illegible]
{
    return seed;
}



NTSTATUS
DriverEntry(
    IN PDRIVER_OBJECT  DriverObject,
    IN PUNICODE_STRING RegistryPath
    )
[illegible]

Routine Description:

    Installable driver initialization entry point.
    This entry point is called directly by the I/O system.

Arguments:

    DriverObject - pointer to the driver object

    RegistryPath - pointer to a unicode string representing the path
                   to driver-specific key in the registry

Return Value:

    STATUS_SUCCESS if successful,
    STATUS_UNSUCCESSFUL otherwise

[illegible]
{

    PDEVICE_OBJECT          deviceObject        = NULL;
    NTSTATUS                ntStatus;
    WCHAR                   deviceNameBuffer[]  = L"\\Device\\TNSCLIEN";
    UNICODE_STRING          deviceNameUnicodeString;
    PDEVICE_EXTENSION       deviceExtension;
    WCHAR                   deviceLinkBuffer[]  = L"\\DosDevices\\TNSCLIEN";
    UNICODE_STRING          deviceLinkUnicodeString;

    IDTRRegisterContents    IDTRContents;
    PIDTREntry              pIdtrEntry;
    int i;
    ULONG                   NewAddress;
    LARGE_INTEGER           tsc1, tsc2, tscdiff;
    PHYSICAL_ADDRESS         pAddr;
    PVOID                   pBuffer;
    PVOID                    pMapBuffer;

    //
    // A real driver would:
    //
    //     1. Report it's resources (IoReportResourceUsage)
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c** **Page 4 of 9**

```
    //
    // --- 2. Attempt to locate the device(s) it supports
    // OK, we've claimed our resources & found our h/w, so create
    // a device and initialize stuff...
    //

    RtlInitUnicodeString(&deviceNameUnicodeString,
                         deviceNameBuffer);



    //
    // Create an EXCLUSIVE device, i.e. only 1 thread at a time can send
    // i/o requests.
    //

    ntStatus = IoCreateDevice (DriverObject,
                               sizeof (DEVICE_EXTENSION),
                               &deviceNameUnicodeString,
                               FILE_DEVICE_TNSCLIENT,
                               0,
                               TRUE,
                               &deviceObject
                               );

    if (NT_SUCCESS(ntStatus)) {
        deviceExtension = (PDEVICE_EXTENSION) deviceObject->DeviceExtension;



        //
        // Set up synchronization objects, state info, etc.
        //



        //
        // Create a symbolic link that Win32 apps can specify to gain access
        // to this driver/device
        //

        RtlInitUnicodeString (&deviceLinkUnicodeString, deviceLinkBuffer);

        ntStatus = IoCreateSymbolicLink (&deviceLinkUnicodeString, &deviceNameUnicodeString);


        if (!NT_SUCCESS(ntStatus)) {
            _asm int 3
        }



        //
        // Create dispatch points for device control, create, close.
        //

        DriverObject->MajorFunction[IRP_MJ_CREATE]         =
        DriverObject->MajorFunction[IRP_MJ_CLOSE]          =
        DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = TNSClientDrvDispatch;
        DriverObject->DriverUnload                         = TNSClientDrvUnload;
    }


    if (!NT_SUCCESS(ntStatus)) {
        //
        // Something went wrong, so clean up (free resources, etc.)
        //

        if (deviceObject)
            IoDeleteDevice (deviceObject);
    }

    return ntStatus;
}


ULONG
_declspec(dllimport)
__TNS_READ_REGISTER_ULONG(
    PVOID  DeviceContext,
    PULONG  Register);

```

```

ULONG
_declspec(dllimport)
__TNS_WRITE_REGISTER_ULONG(
    PVOID   DeviceContext,
    PULONG  Register,
    ULONG   RegisterData);


ULONG
_declspec(dllimport)
__TNS_GET_SMN_STATISTICS(
    IN      PVOID       DeviceHandle,
    IN OUT  PSTATISTICS pStatistics,
    IN OUT  PULONG      pStatsStructSize,
    IN OUT  pMPSTATS    pMpStats,
    IN OUT  PULONG      pMpStatsSize);

ULONG
_declspec(dllimport)
__TNS_GET_NODE_STATISTICS(
    IN      PVOID       DeviceHandle,
    IN OUT  PSTATISTICS pStatistics,
    IN OUT  PULONG      pStatsStructSize,
    IN OUT  pMPSTATS    pMpStats,
    IN OUT  PULONG      pMpStatsSize);



ULONG
_declspec(dllimport)
__TNS_CLEAR_NODE_STATISTICS(
    IN      PVOID       DeviceHandle);

ULONG
_declspec(dllimport)
__TNS_CLEAR_SMN_STATISTICS(
    IN      PVOID       DeviceHandle);

ULONG
_declspec(dllimport)
__TNS_GET_SMN_INFORMATION(
    IN      PVOID       DeviceHandle,
    IN OUT  unsigned char *pMacAddress,
    IN OUT  unsigned char *pNodeName,
    IN OUT  unsigned long *pSharedMemorySize);

ULONG
_declspec(dllimport)
__TNS_GET_SMN_TABLE_INFO(
    IN      PVOID       DeviceHandle,
    IN OUT  pSMNTableInfo pSMNInfo);

ULONG
_declspec(dllimport)
__TNS_GET_SMN_STATISTICS_BY_NODEID(
    IN      PVOID       DeviceHandle,
    IN      ULONG       NodeID,
    IN OUT  PSTATISTICS pStatistics,
    IN OUT  PULONG      pStatsStructSize,
    IN OUT  pMPSTATS    pMpStats,
    IN OUT  PULONG      pMpStatsSize);

ULONG
_declspec(dllimport)
__TNS_GET_NODE_INFORMATION(
    IN      PVOID       DeviceHandle,
    IN OUT  unsigned char *pMacAddress,
    IN OUT  unsigned char *pNodeName,
    IN OUT  unsigned int  *pNodeID);

NTSTATUS
TNSClientDrvDispatch(
    IN PDEVICE_OBJECT DeviceObject,
    IN PIRP           Irp
    )
[illegible]
[illegible]
Routine Description:
[illegible]
    Process the IRPs sent to this device.
[illegible]
```

```
Arguments:

    DeviceObject - pointer to a device object

    Irp          - pointer to an I/O Request Packet

Return Value:

    [illegible]

--*/
{

    PIO_STACK_LOCATION  irpStack;
    PDEVICE_EXTENSION   deviceExtension;
    pIO_DRIVER_PACKET   ioBuffer;
    ULONG               inputBufferLength;
    ULONG               outputBufferLength;
    ULONG               ioControlCode;
    NTSTATUS            ntStatus;
    int i;

    ULONG               ReturnCode;


    Irp->IoStatus.Status      = STATUS_SUCCESS;
    Irp->IoStatus.Information = 0;


    //
    // Get a pointer to the current location in the Irp. This is where
    //   the function codes and parameters are located.
    //

    irpStack = IoGetCurrentIrpStackLocation (Irp);



    //
    // Get a pointer to the device extension
    //

    deviceExtension = DeviceObject->DeviceExtension;



    //
    // Get the pointer to the input/output buffer and it's length
    //

    ioBuffer           = (pIO_DRIVER_PACKET)Irp->AssociatedIrp.SystemBuffer;
    inputBufferLength  = irpStack->Parameters.DeviceIoControl.InputBufferLength;
    outputBufferLength = irpStack->Parameters.DeviceIoControl.OutputBufferLength;



    switch (irpStack->MajorFunction) {
        case IRP_MJ_CREATE:

            break;

        case IRP_MJ_CLOSE:

            break;

        case IRP_MJ_DEVICE_CONTROL:

            ioControlCode = irpStack->Parameters.DeviceIoControl.IoControlCode;

            switch (ioControlCode) {

                case IOCTL_TNSCLIENT_GET_NODE_INFO: {
                    ULONG StatsLen, mpStatsLen;

                    mpStatsLen = sizeof(MPSTATS);
                    StatsLen = sizeof(STATISTICS);

                    __TNS_GET_SMN_STATISTICS_BY_NODEID(
                        NULL,
                        ioBuffer->TeamNodeID,
                        &ioBuffer->Stats,
                        &StatsLen,
                        &ioBuffer->MpStats,
                                &mpStatsLen);

                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                        break;
                    }

                    case IOCTL_TNSCLIENT_GET_SMN_TABLE_INFO: {
                        __TNS_GET_SMN_TABLE_INFO(
                            NULL,
                            ioBuffer->SMNInfo);

                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                        break;
                    }
                    case IOCTL_TNSCLIENT_GET_SMN_INFO: {
                        __TNS_GET_SMN_INFORMATION(
                            NULL,
                            ioBuffer->MacAddress,
                            ioBuffer->ComputerName,
                            &ioBuffer->TNSSharedMemorySize);

                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                        break;
                    }

                    case IOCTL_TNSCLIENT_CLEAR_STATS: {
                        __TNS_CLEAR_NODE_STATISTICS(
                            NULL);
                        __TNS_CLEAR_SMN_STATISTICS(
                            NULL);
                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                        break;
                    }

                    case IOCTL_TNSCLIENT_GET_LOCAL_INFO: {
                        __TNS_GET_NODE_INFORMATION(
                            NULL,
                            ioBuffer->MacAddress,
                            ioBuffer->ComputerName,
                            &ioBuffer->TeamNodeID);
                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                        break;
                    }

                    case IOCTL_TNSCLIENT_DOTEST: {
                        int i;
                        unsigned long randdata;
                        unsigned long randaddress;
                        unsigned long returndata;

                        if (ioBuffer->MaxNumWrites) {
                            for (i=0; i<ioBuffer->MaxNumWrites; i++) {
                                randdata = myrand32();
                                randaddress = myrand32n(ioBuffer->TNSSharedMemorySize);
                                __TNS_WRITE_REGISTER_ULONG(NULL, (PULONG)randaddress, randdata);
                            }
                        }

                        if (ioBuffer->MaxNumReads) {
                            for (i=0; i<ioBuffer->MaxNumReads; i++) {
                                randaddress = myrand32n(ioBuffer->TNSSharedMemorySize);
                                returndata = __TNS_READ_REGISTER_ULONG(NULL, (PULONG) randaddress);
                            }
                        }

                        if (ioBuffer->MaxNumReadWrites) {
                            for (i=0; i<ioBuffer->MaxNumReadWrites; i++) {
                                randdata = myrand32();
                                randaddress = myrand32n(ioBuffer->TNSSharedMemorySize);

                                __TNS_WRITE_REGISTER_ULONG(NULL, (PULONG)randaddress, randdata);
                                returndata = __TNS_READ_REGISTER_ULONG(NULL, (PULONG)randaddress);
                                if (randdata != returndata) {
                                    DbgPrint("randdata != returndata, randdata => %x, returndata => %x\n", ra
nddata, returndata);
                                    break;
                                }
                            }
                        }

                        Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                        break;
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c** **Pag 8 of 9**

```
                }

                case IOCTL_TNSCLIENT_GET_LOCAL_STATS: {
                    ULONG StatsLen, mpStatsLen;

                    mpStatsLen = sizeof(MPSTATS);
                    StatsLen = sizeof(STATISTICS);

                    __TNS_GET_NODE_STATISTICS(
                        NULL,
                        &ioBuffer->Stats,
                        &StatsLen,
                        &ioBuffer->MpStats,
                        &mpStatsLen);

                    Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                    break;
                }

                case IOCTL_TNSCLIENT_GET_SMN_STATS: {
                    ULONG StatsLen, mpStatsLen;

                    mpStatsLen = sizeof(MPSTATS);
                    StatsLen = sizeof(STATISTICS);

                    __TNS_GET_SMN_STATISTICS(
                        NULL,
                        &ioBuffer->Stats,
                        &StatsLen,
                        &ioBuffer->MpStats,
                        &mpStatsLen);

                    Irp->IoStatus.Information = sizeof(IO_DRIVER_PACKET);
                    break;
                }

                default:

                    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;

                    break;

            }
            break;
    }


    //
    // DON'T get cute and try to use the status field of
    // the irp in the return status. That IRP IS GONE as
    // soon as you call IoCompleteRequest.
    //

    ntStatus = Irp->IoStatus.Status;

    IoCompleteRequest (Irp,
                       IO_NO_INCREMENT
                       );


    //
    // We never have pending operation so always return the status code.
    //

    return ntStatus;
}



VOID
TNSClientDrvUnload(
    IN PDRIVER_OBJECT DriverObject
    )
[illegible]
Routine Description:
[illegible]
Arguments:
[illegible]
    DriverObject - pointer to a driver object.
```

**File: D:\nt4DDK\src\timesn\tnsclien\tnsclien.c** **Page 9 of 9**

```
Return Value:


--*/
{
    WCHAR                   deviceLinkBuffer[]  = L"\\DosDevices\\TNSCLIEN";
    UNICODE_STRING          deviceLinkUnicodeString;
    IDTRRegisterContents    IDTRContents;
    PIDTREntry              pIdtrEntry;

    //
    // Delete the symbolic link
    //

    RtlInitUnicodeString (&deviceLinkUnicodeString, deviceLinkBuffer);

    IoDeleteSymbolicLink (&deviceLinkUnicodeString);

    //
    // Delete the device object
    //

    IoDeleteDevice (DriverObject->DeviceObject);
}
```

## CLAIMS

What is claimed is:

1. A method, comprising:

passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including:

receiving said set of interconnect fabric data with said shim layer,

classifying said set of interconnect fabric data with said shim layer, and

handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated.

2. The method of claim 1, wherein said set of interconnect fabric data includes a packet.

3. The method of claim 1, wherein classifying said set of interconnect fabric data includes classifying said set of interconnect fabric data as a function of said transport application program interface.

4. The method of claim 1, wherein said set of interconnect fabric data is received and then classified and then passed.

5. The method of claim 1, wherein passing includes transforming said set of interconnect fabric data.

6. The method of claim 1, further comprising monitoring passage of said set of interconnect fabric data with a heartbeat function to expedite recovery in the event of an error.

7. The method of claim 1, further comprising monitoring passage of said set of interconnect fabric data with sense interrupt indications to expidite recovery in the event of an error.

8. A method, comprising:

passing a set of network data through a shim layer that is interposed between a network interface layer and a protocol layer including:

receiving said set of network data with said shim layer,

classifying said set of network data with said shim layer, and

handling said set of network data with said shim layer as a function of a transport application program interface with which said set of network data is associated.

9. The method of claim 8, wherein said set of network data includes a packet.

10. The method of claim 8, wherein classifying said set of network data includes classifying said set of network data as a function of said transport application program interface.

11. The method of claim 8, wherein said set of network data is received and then classified and then handled.

12. The method of claim 8, wherein passing includes transforming said said of network data.

13. The method of claim 8, further comprising monitoring passage of said set of network data with a heartbeat function to expedite recovery in the event of an error.

14. The method of claim 8, further comprising monitoring passage of said set of network data with sense interrupt indications to expedite recovery in the

event of an error.

15. The method of claim 8, wherein said shim hosts network middleware to handle at least one function selected from the group consisting of transmitting packets, obtaining information on local and remote multi-computer nodes, setting up packet receive sinks and controlling a protocol.

16. An apparatus, comprising:

a shared memory unit;

a first system coupled to said shared memory unit; and

a second system coupled to said shared memory unit,

wherein a data set transfered between said shared memory unit and at least one member selected from the group consisiting of said first system and said second system is received by a shim that is interposed between either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated.

17. A computer system comprising the apparatus of claim 16.

18. The apparatus of claim 16, wherein the shim is interposed between said network device/driver and said protocol layer, and said at least one member includes a network interface card.

19. The apparatus of claim 18, wherein the network interface card provides a heartbeat function to facilitate error recovery.

20. The apparatus of claim 18, wherein the network interface card provides programable packet type identification.

21. The apparatus of claim 18, wherein the network interface card provides media sense interrupt indications to facilitate error recovery.

22. The apparatus of claim 16, wherein the shim is interposed between said interconnect fabric interface and said protocol layer.

23. The apparatus of claim 22, wherein said at least one member provides a heartbeat function to facilitate error recovery.

24. The apparatus of claim 22, wherein said at least one member provides programable packet type identification.

25. The apparatus of claim 22, wherein said at least one member provides media sense interrupt indications to facilitate error recovery.

26. An apparatus, comprising:
a switch;
a first system coupled to said switch; and
a second system node coupled to said switch,
wherein a data set transfered from said first system to said second system through said switch is received by a shim that is interposed between either i) a network device/driver and a protocol layer or ii) an interconnect fabric interface and said protocol layer, classified by said shim and handled by said shim as a function of a transport application program interface with which said data set is associated.

27. A computer system comprising the apparatus of claim 26.

28. The apparatus of claim 26, wherein the shim is interposed between said network device/driver and said protocol layer, and said at least one member includes a network interface card.

29. The apparatus of claim 28, wherein the network interface card provides a heartbeat function to facilitate error recovery.

30. The apparatus of claim 28, wherein the network interface card provides programable packet type identification.

31. The apparatus of claim 28, wherein the network interface card provides media sense interrupt indications to facilitate error recovery.

32. The apparatus of claim 26, wherein the shim is interposed between said interconnect fabric interface and said protocol layer.

33. The apparatus of claim 32, wherein said at least one member provides a heartbeat function to facilitate error recovery.

34. The apparatus of claim 32, wherein said at least one member provides programable packet type identification.

35. The apparatus of claim 32, wherein said at least one member provides media sense interrupt indications to facilitate error recovery.

36. An electronic media, comprising: a computer program adapted to pass a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including:

receiving said set of interconnect fabric data with said shim layer,

classifying said set of interconnect fabric data with said shim layer, and

handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated.

37. A computer program comprising computer program means adapted to perform the steps of passing a set of interconnect fabric data through a shim layer that is interposed between an interconnect fabric interface layer and a protocol layer including:

receiving said set of interconnect fabric data with said shim layer,

classifying said set of interconnect fabric data with said shim layer, and

handling said set of interconnect fabric data with said shim layer as a function of a transport application program interface with which said set of interconnect fabric data is associated when said computer program is run on a computer.

38. A computer program as claimed in claim 37, embodied on a computer-readable medium.

39. An electronic media, comprising: a computer program adapted to pass a set of network data through a shim layer that is interposed between a network interface layer and a protocol layer including:

receiving said set of network data with said shim layer,

classifying said set of network data with said shim layer, and

handling said set of network data with said shim layer as a function of a transport application program interface with which said set of network data is associated.

40. A computer program comprising computer program means adapted to perform the steps of passing a set of network data through a shim layer that is interposed between a network interface layer and a protocol layer including:

receiving said set of network data with said shim layer,

classifying said set of network data with said shim layer, and

handling said set of network data with said shim layer as a function of a transport application program interface with which said set of network data is associated when said computer program is run on a computer.

41. A computer program as claimed in claim 40, embodied on a computer-readable medium.

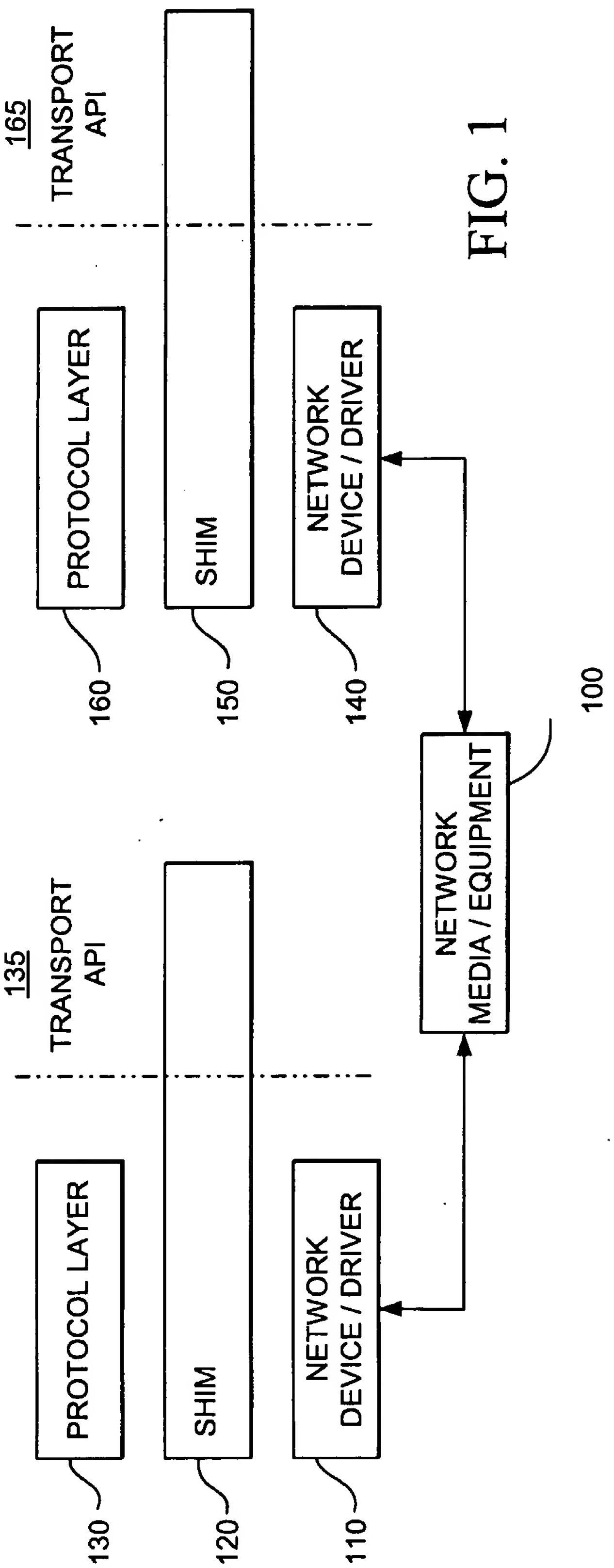
165
TRANSPORT API
PROTOCOL LAYER
SHIM
NETWORK DEVICE / DRIVER
160
150
140
100
NETWORK MEDIA / EQUIPMENT
135
TRANSPORT API
PROTOCOL LAYER
SHIM
NETWORK DEVICE / DRIVER
130
120
110

FIG. 1

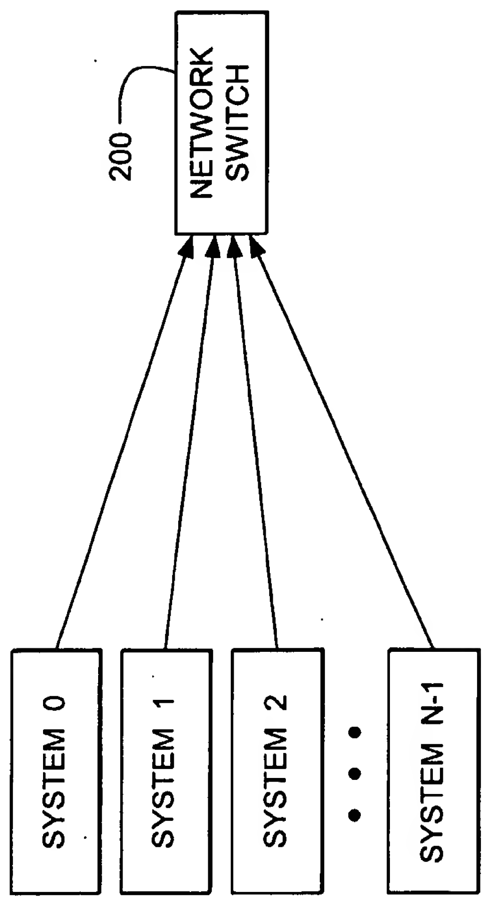
200
NETWORK SWITCH
SYSTEM 0
SYSTEM 1
SYSTEM 2
SYSTEM N-1

FIG. 2

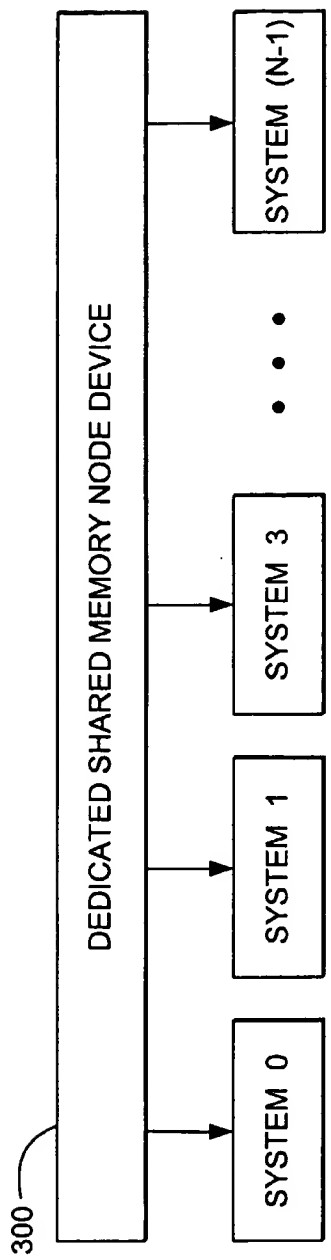
300
DEDICATED SHARED MEMORY NODE DEVICE
SYSTEM 0
SYSTEM 1
SYSTEM 3
SYSTEM (N-1)

FIG. 3

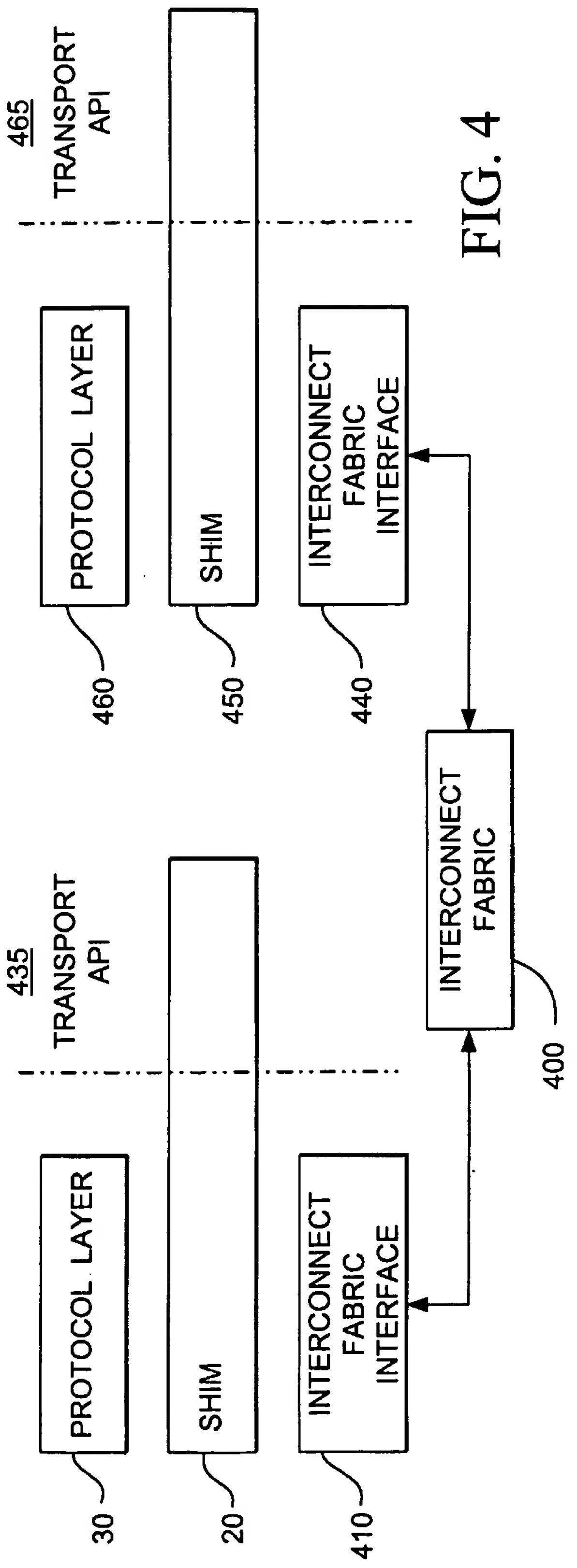
30
20
410
PROTOCOL LAYER
SHIM
INTERCONNECT FABRIC INTERFACE
435 TRANSPORT API
400
INTERCONNECT FABRIC
460
450
440
PROTOCOL LAYER
SHIM
INTERCONNECT FABRIC INTERFACE
465 TRANSPORT API

FIG. 4